Latency-aware Resource Management at the Edge

Klervie Toczé
This is a Swedish Licentiate’s Thesis

Swedish postgraduate education leads to a doctor’s degree and/or a licentiate’s degree. A doctor’s degree comprises 240 ECTS credits (4 years of full-time studies). A licentiate’s degree comprises 120 ECTS credits.

Edition 1:1

© Klervie Toczé, 2020
ISBN 978-91-7929-904-0
ISSN 0280-7971
URL http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-163388

Published articles have been reprinted with permission from the respective copyright holder.
Typeset using LATEX

Printed by LiU-Tryck, Linköping 2020
The increasing diversity of connected devices leads to new application domains being envisioned. Some of these need ultra low latency or have privacy requirements that cannot be satisfied by the current cloud. By bringing resources closer to the end user, the recent edge computing paradigm aims to enable such applications.

One critical aspect to ensure the successful deployment of the edge computing paradigm is efficient resource management. Indeed, obtaining the needed resources is crucial for the applications using the edge, but the resource picture of this paradigm is complex. First, as opposed to the nearly infinite resources provided by the cloud, the edge devices have finite resources. Moreover, different resource types are required depending on the applications and the devices supplying those resources are very heterogeneous. This thesis studies several challenges towards enabling efficient resource management for edge computing.

The thesis begins by a review of the state-of-the-art research focusing on resource management in the edge computing context. A taxonomy is proposed for providing an overview of the current research and identify areas in need of further work.

One of the identified challenges is studying the resource supply organization in the case where a mix of mobile and stationary devices is used to provide the edge resources. The ORCH framework is proposed as a means to orchestrate this edge device mix. The evaluation performed in a simulator shows that this combination of devices enables higher quality of service for latency-critical tasks.

Another area is understanding the resource demand side. The thesis presents a study of the workload of a killer application for edge computing: mixed reality. The MR-Leo prototype is designed and used as a vehicle to understand the end-to-end latency, the throughput, and the characteristics of the workload for this type of application. A method for modeling the workload of an application is devised and applied to MR-Leo in order to obtain a synthetic workload exhibiting the same characteristics, which can be used in further studies.

This work has been supported by the Swedish National Graduate School in Computer Science (CUGS).
Acknowledgments

First, I would like to thank my main supervisor Simin Nadjm-Tehrani for our discussions and her guidance that have made me progress in my research. I also want to thank my second supervisor Nahid Shahmehri for her additional advice and support.

Then I want to thank all the members of RTSLAB, past or present, for contributing to a nice working environment with inspiring discussions at our meetings or during the fikas. Thanks to Mikaela, Åsa, Lene and Anne for their help when administrative issues or questions arose during the past three years.

Thanks as well to all the persons I met at conferences and in other research contexts for interesting discussions. The insights gained from those occasions contributed to shaping my work and this thesis.

A big thank you to our SaS lunch group for providing relaxing breaks that can both provide a well-needed distraction away from work but also useful insights from other research fields. Thanks to everyone involved in the IDA PhD council, LiUPhD, and related groups, I truly believe that our work there has a positive impact on the study life at LiU. Special thanks to Adrian, Adrian, Antonia, Arian, Felipe, Flavia, Hannah, Mikaela, Rodrigo, Sarah, Viktor, and Ulf for making my PhD time enjoyable even outside the university. Tack Blåset och den bästa phloytsektionen for introducing me to Swedish student life. Finally, to all my friends from further away, thank you for your understanding when I am talking about work on vacation.

I also want to thank my family that always supported and encouraged me through my studies and life in general. I am immensely grateful for this.

Last but not least, a huge thank you to my husband Michäel for his unconditional support, and for going on (more or less) epic quests with me. I look forward to experiencing with you the adventures that lie ahead of us.

Klervie Toczé
Linköping, Sweden
February 2020
# Contents

Abstract iii  
Acknowledgments v  
Contents vii  
List of Figures ix  
List of Tables xi  

1 Introduction 1  
1.1 Edge Computing 1  
1.2 Motivation 4  
1.3 Problem Formulation 5  
1.4 Contributions 5  
1.5 List of Publications 6  
1.6 Thesis Outline 6  

2 Background 9  
2.1 Edge Computing Architectures 9  
2.2 Mixed Reality 13  
2.3 Video Streaming 16  

3 Resource Management at the Edge 19  
3.1 Overview 19  
3.2 Taxonomy 22  
3.3 Resource Type 22  
3.4 Objective 25  
3.5 Resource Location 32  
3.6 Resource Use 33  

4 Distributed Orchestration using Mobile Edge Devices 39  
4.1 D2TEP Problem 39  
4.2 Considered Scenario 40
# List of Figures

2.1 Edge server architecture. ........................................ 11
2.2 Coordinator device architecture. ................................. 12
2.3 Device cloud architecture. ...................................... 13
2.4 Reality-virtuality continuum. .................................. 14
2.5 MR examples. ..................................................... 14
2.6 MR application pipeline ........................................ 15

3.1 Distribution of the selected papers according to their architectural category. ........................................ 21
3.2 Generic idea of edge offloading. ................................. 22
3.3 A taxonomy of resource management at the edge. ............. 23

4.1 System model considered for the ORCH framework. .......... 41
4.2 Example of division of a neighborhood into segments. ........ 42
4.3 Overview of the ORCH framework. .............................. 46
4.4 Schematic view of the considered neighborhood. ............. 56
4.5 Impact of task placement strategy (balanced load). .......... 58
4.6 Impact of task placement strategy (overload). ................. 59
4.7 Success ratios for DS tasks (balanced load). .................. 60
4.8 Success ratios (balanced load). ................................ 61
4.9 Division of completed tasks among mobile/stationary edge devices (balanced load). ................................. 61
4.10 Success ratios for DS tasks (overload). ......................... 62
4.11 Success ratios (balanced load). ................................ 62
4.12 Division of completed tasks among mobile/stationary edge devices (overload). ........................................ 63

5.1 Edge offloading for our MR case study. ....................... 66
5.2 MR workflow for video input. .................................. 67
5.3 MR workflow for user input. ................................... 67
5.4 Overview of the MR-Leo architecture. ......................... 68
5.5 Screenshot of the MR-Leo client ............................... 69
5.6 Screenshot of the test video. ................................... 71
5.7 Test environment .................................................. 73
5.8 Latency CDFs for the baseline configuration ................. 74
5.9 CDF of the throughput for the baseline configuration ....... 75
5.10 Latency breakdown for different configurations ............. 77
5.11 Number of tracked points depending on transport protocol and bandwidth ................................................. 79
5.12 Latency CDFs for the UDP configuration ..................... 79
5.13 Throughput CDFs for different configurations ............... 80
5.14 Latency CDFs for the MJPEG configuration ................... 80
6.1 Overview of the characterization and modeling workflow. ... 88
6.2 Experimental frame arrival rate at the edge device .......... 92
6.3 Measured CPU use per second ................................. 95
6.4 Measured number of instructions per frame for the reference video ......................................................... 96
6.5 Density function of the fitted distribution ..................... 97
6.6 QQ-plot of the data vs the fitted distribution ................. 97
6.7 Measured number of instructions per frame .................. 98
6.8 Measured number of instructions per frame .................. 99
6.9 Histogram of the memory usage of the MR application .... 100
6.10 Interactive MR application model .............................. 103
6.11 Task arrival times for one sample of generated tasks using different models ............................................ 104
6.12 Density function of the number of instructions per task in the Sonmez model ............................................ 105
List of Tables

3.1 Selected articles according to objective of resource management. . . 26
3.2 Applications considered in reviewed articles. . . . . . . . . . . . . 35

4.1 Simulation parameters. . . . . . . . . . . . . . . . . . . . . . . . . . 57
4.2 Average, standard deviation, minimum, and maximum number of successful DS tasks for the balanced load (B) and overload (O) scenarios. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60

5.1 Breakdown of the average FRTT. . . . . . . . . . . . . . . . . . . . . . 76
5.2 Performance measurements statistics (ms). . . . . . . . . . . . . . 77

6.1 Load indicators considered in selected works using simulation. . . 87
6.2 Basic statistics for the instruction count (MI). . . . . . . . . . . . . . 95
6.3 Overview of the proposed MR workload model. . . . . . . . . . . 103
The need for computational power is shaping today’s society. More and more domains of our daily lives are *smartified* and new services emerge. In order to achieve this, there is a need for *gathering* data and *analyzing* it. Moreover, new envisioned services require ultra low latency and there are increasing concerns about end users privacy. In recent years, a new paradigm has emerged in order to tackle these challenges. This is the **edge computing** paradigm.

### 1.1 Edge Computing

Since the advent of computing and IT systems, several paradigms have succeeded each other, as the technology and needs evolve. This section details how the edge computing paradigm considered in this thesis appeared and what its characteristics are.

#### 1.1.1 Origins

There is an identified trend in IT systems that consists in alternating between centralization and decentralization [1]. The latest widely adopted paradigm is *cloud computing*, a heavily centralized paradigm where tasks are sent to remote huge datacenters that gather a lot of servers used for computation or storage purposes.

This paradigm has been very successful since the computing power available in the cloud is without comparison to the one available in end devices. However, more and more applications want to use the cloud and this means that the network traffic is increasing to a point that it will not be manageable with the current infrastructure [2]. Moreover, some applications require ultra
low latency, which the cloud is not able to provide, so there is a push from
the cloud area to get a solution to reduce the load on the network and offer
lower response times [3].

At the same time, the number of connected devices is increasing, since
the Internet of Things (IoT) is in an expansion phase. According to the lat-
est Ericsson mobility report [4], it is expected that 22.3 billion of IoT devices
will be connected in 2024, in addition to 7.2 billion smartphones. This can be
compared to the 8.6 billion IoT devices and 5.1 billion smartphones actually
connected in 2018. All kind of devices (temperature sensors, fridges, lights,
etc.) can now be seen as data producers and a new type of computation ca-
pability is required for handling the tremendous amount of data generated.
Using the cloud is not a single alternative any more because of the additional
load on the network this would create, but also due to growing privacy con-
cerns and energy constraints [3].

The edge computing paradigm promises to provide for both those emerging
needs, complementing cloud computing.

1.1.2 Concept and Aims

The main concept behind the edge computing paradigm consists in having
network nodes with computational and storage resources close to the user
devices (mobile phones, sensors), at the edge of the current network. Those
network nodes form an intermediate level between the traditional cloud and
the user devices.

Edge computing is an innovative area bringing together diverse business
sectors such as telecommunication actors, vehicle vendors, cloud providers,
and emerging application or device providers. Therefore, the terminology
used in research works for naming the above concept is diverse and still
evolving.

Several of the used denominations are: mobile cloud computing [5], fog
computing [6], edge computing [7], mobile edge computing [8], path com-
puting [9], mobile edge cloud [10], mobile edge network [11], follow-me
cloud [12], mobile follow-me cloud [13], multi-tier cloud federations [14],
small cell cloud [15], fast moving personal cloud [16], CONCERT [17], dis-
tributed clouds [18], and femtoclouds [19, 20].

The common aim behind the above denominations is to provide services
with high responsiveness, enable scalability and the enforcement of privacy
policies, and mitigate cloud outages [7]. In this thesis, we choose to use the
denomination edge computing for denoting this concept.
1.1. Edge Computing

1.1.3 Focus Areas

As the body of research work about edge computing is increasing, review studies provide an overview of the area. We present the different focus areas of edge computing in order to situate the thesis work in this landscape.

Early on, in 2012, Fernando et al. [5] studied mobile cloud computing research, which included mobile devices connecting to a cloud, mobile devices collaborating with each other, and mobile devices connecting to a local cloud. The last two variants are two architecture variants of the edge paradigm. They propose a taxonomy of the issues encountered within mobile cloud computing and how they had been addressed at that time.

As edge computing takes place at a layer in between the devices and the cloud, researchers have studied it with different perspectives, depending on their primary interest area. Shi et al. [3] describe the edge paradigm and explain its dynamic, at the confluence of IoT and cloud computing areas. In particular, IoT challenges are presented by Chiang and Zhang [21]. They describe how those require a new architecture that the edge computing paradigm can provide. Then, they present use cases and research challenges, but do not name resource management as one of them.

When presenting a recent (2018) overview of mobile edge computing, Abbas et al. [22] present several applications as well as research ongoing in the areas of computational offloading, low latency, storage, and energy efficiency, that they identified as being the ones where most work has been conducted. They also present efforts in creating testbeds and works related to security and privacy issues. According to them, resource optimization is one currently open issue.

The takeaway from studies on the edge computing paradigm as a whole is that, although resource management was not one of the main research areas at an earlier stage, it has become a more important focus area in the last few years. This serve as a first motivation for the thesis. The thesis provides a more up to date picture of edge computing resource management research.

Apart from resource management, there are other focus areas in edge computing. One of them is edge computing deployment, which presents several technological challenges. Liu et al. [10] present architectures and frameworks that were used for designing both edge servers and the network between the different devices involved. Another area is security, e.g. as studied by Roman et al. [8]. In their work, they describe the security threats that exist within the edge paradigm and review state-of-the-art research.

In parallel to those focus areas, it is complementary to look at edge computing from an edge application area, or a technology perspective. Examples of such overviews can be found in the case of smart cities [23], augmented reality [24], software-defined networking [25], or blockchain deployments [26]. This thesis is focusing on the area of resource management and other edge computing areas are out of the scope.
1. INTRODUCTION

1.2 Motivation

In order to make edge computing a reality (and a success), the area of resource management is critical for an obvious reason: without reliable access to edge resources, no edge service can be provided with a reasonable quality of service. Moreover, even though edge computing has some similarities with cloud computing, they are very different when considering the resource perspective. Indeed, the resource aspect of edge computing is more complex. For example, resource suppliers are highly spread out within the area to be serviced, the amount of resource provided by each edge device is small compared to a cloud datacenter, and the devices providing the edge resources are very heterogeneous. This creates some research challenges specific for the edge computing paradigm.

Resource management for edge computing itself is a wide area, with an increasing body of research work. It is therefore currently hard to get an overall view about this area. Getting this overall view is crucial in order for the work included in this thesis to be relevant.

One part of resource management is to organize the resource supply. Among other aspects, it means that an edge infrastructure has to be deployed. The resource supply organization has to be performed in an efficient manner in order to avoid waste of resources while providing high quality of service. As the geographical areas concerned are large, the traditional way of providing resources, i.e. by installing stationary devices at selected locations, is not going to be sufficient. This is due to lack of flexibility in dealing with sudden changes in the locality of the arriving load. Therefore, alternative ways of providing resources should be envisioned and studied. One of those alternatives is provide resources with a mix of stationary and mobile devices.

The other part of resource management is about estimating the resource demand. This part is of the highest importance for ensuring the success of edge computing. Indeed, highly-advanced resource supply technologies are of no use if they are unable to satisfy the resource demand. Therefore, there is also a need to study the specificity of edge computing applications, which will create the resource demand on the edge infrastructure. Certain applications areas for edge computing have been identified in the literature, such as augmented reality [27, 22]. However, a study of actual edge applications is not possible at the moment due to the absence of available edge-managed ones. Therefore, there is a need for generic methods which can be applied to several forthcoming edge applications.

Thus, it is a good start to have available edge applications that can run on prospective edge devices in order to study resource management techniques. However, this is not enough. Indeed, the edge computing paradigm includes a very large number of devices, both for the demand and the supply side. Therefore, the edge infrastructure and algorithms need to scale very
well to handle the incoming workload. This is so far not possible to study on an actual infrastructure as real large-scale edge deployments do not exist. Instead, researchers use simulation tools for conducting such large scale studies. So far, the load used as an input for those studies is theoretical, due to the lack of real edge applications. When those are developed, there is a need to be able to extract their characteristics into a model to later generate simulated load that emulates the actual application load.

1.3 Problem Formulation

Motivated by these considerations, three problems are formulated for studying in this thesis:

- Study the state-of-the-art in research about the new area of edge computing resource management to identify research gaps.
- Devise and evaluate ways of organizing the edge resource supply to combine providing high quality of service and using resources efficiently.
- Investigate and analyze workloads created by applications using edge computing to understand their specific characteristics and the associated resource demand.

1.4 Contributions

This thesis contributes to resource management advances for edge computing by providing elements of answers to the three problems formulated above. The thesis proposes an overview of the area and is a step towards providing efficient resource management in the context of edge computing.

More precisely, the contributions of this thesis are the following:

1. A review of state-of-the-art edge resource management research: A novel taxonomy of edge resource management is proposed based on edge computing research and remaining challenges.

2. A framework for edge orchestration: The ORCH framework is proposed for managing a combination of mobile and stationary edge resources. Moreover, a proof-of-concept implementation and evaluation of the ORCH framework in a simulator is presented.

3. A study and modeling of an edge application prototype: First, a generic architecture for offloading analysis of video streams to the edge and a prototype mixed reality implementation, MR-Leo, is presented. In order to gain understanding of the demand of this type of application and identify performance bottlenecks, throughput and end-to-end
1. Introduction

latency are analyzed for different component choices. Then, a generic method for characterizing and modeling edge workloads based on real applications is devised. The method is applied to the MR-Leo prototype in order to obtain one mixed reality edge workload model as an input to future simulations.

1.5 List of Publications

The work presented in this thesis is based on the following publications:

- K. Toczé and S. Nadjm-Tehrani, *Where Resources meet at the Edge*, in *Proceedings of IEEE International Workshop on Secure and Resource-Efficient Edge Computing 2017, held in conjunction with CIT*, IEEE, 2017.

- K. Toczé and S. Nadjm-Tehrani, *A Taxonomy for Management and Optimization of Multiple Resources in Edge Computing*, *Wireless Communications and Mobile Computing*, Wiley-Hindawi, 2018.

- K. Toczé and S. Nadjm-Tehrani, *ORCH: Distributed Orchestration Framework using Mobile Edge Devices*, in *Proceedings of the 3rd IEEE International Conference on Fog and Edge Computing (ICFEC 2019)*, IEEE, 2019.

- K. Toczé, N. Schmitt, I. Brandic, A. Aral, and S. Nadjm-Tehrani, *Towards Edge Benchmarking: A Methodology for Characterizing Edge Workloads*, in *Proceedings of the 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, IEEE, 2019.

- K. Toczé, J. Lindqvist, and S. Nadjm-Tehrani, *Performance Study of Mixed Reality for Edge Computing*, in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing (UCC2019)*, ACM, 2019. Winner of the best paper award.

The last article above is a work building on the following master thesis, which was supervised by K. Toczé and examined by S. Nadjm-Tehrani:

- J. Lindqvist, *Edge Computing for Mixed Reality*, Master’s thesis, Linköping University, 2019.

1.6 Thesis Outline

The rest of the thesis is organized as follows:

*Chapter 2* introduces background knowledge required for reading the thesis. This comprises a description of edge computing architectures, mixed reality, and video streaming.
Chapter 3 gives an overview of the resource management area for edge computing by presenting a novel taxonomy together with a state-of-the-art review.

Chapter 4 studies one alternative for supplying resources at the edge by investigating the benefits of using a combination of stationary and mobile edge devices in the edge infrastructure. In order to manage the edge devices, a distributed framework is proposed and its implementation evaluated in a simulator.

Chapter 5 dives into the resource demand part and presents the performance study of a mixed reality edge application prototype, MR-Leo. This open-source prototype is modular and several configurations are investigated in order to understand throughput and latency.

Chapter 6 continues with the resource demand part by introducing a method for creating an abstract/generic model of edge workloads based on available applications. The method is then applied to MR-Leo.

Chapter 7 concludes this thesis and presents directions for future work.
This chapter presents the background knowledge required to understand the rest of the work presented in this thesis. Section 2.1 gives an overview of the edge computing architectures, including a description of the devices involved. Then, Section 2.2 presents the application domain used as a case study in Chapter 5 and 6, which is mixed reality. In particular, the case study uses video streaming, which is introduced in Section 2.3.

2.1 Edge Computing Architectures

As the edge computing paradigm is emerging at the confluence of already existing paradigms and concepts, those are shaping the different architectural variants that edge computing can use.

2.1.1 Devices Involved

Following the development of the IoT, it is nowadays not only computers or smartphones which can be connected to the network, but a large variety of things such as cars, sensors, drones, robots, or home appliances. All those objects located at the user end of the network, which produce data or need cloud/edge resources are called end devices in the thesis.

Devices installed close to the end user specifically for edge computing purposes are called edge devices. Under this term are also included the devices that are already now connecting the end devices to the rest of the network, for example, home routers, gateways, access points, or base stations, which are becoming increasingly powerful [28].
Finally, physical components of the cloud are referred to by the term *cloud devices*. Those network device classifications are used to create three different *levels* in the network: the device level, the edge level, and the cloud level.

### 2.1.2 Current Status of Edge Architectures

Industry and research initiatives such as the Open Edge Computing\(^1\) community, the OpenFog Consortium (that merged with the Industrial Internet Consortium in early 2019)\(^2\), and a European Telecommunications Standards Institute (ETSI) standardization group working on Multi-access Edge Computing\(^3\) have been working on defining standards for the edge computing paradigm. Standardization efforts coming from the ETSI group have been reviewed in detail by Mao et al. [29] and Mach et al. [30]. This group is regularly publishing specifications for different aspects of Multi-Access Edge Computing, including a reference architecture [31] in January 2019. Mao et al. [29] also present edge standardization efforts within the 5G standard. In June 2018, the IEEE Standards Association adopted the reference architecture from the OpenFog Consortium as a standard for fog computing [32].

Even though some standards are emerging, current research on edge computing is using several different architectures and there is ongoing work for defining edge computing architectures. Recent surveys focus on presenting these architectures. For example, Liu et al. [10] review different architectures for Mobile Edge Cloud servers and networks, and Mach et al. [30] present an overview of proposed solutions enabling computation to be brought close to the end device within the field of mobile edge computing. The approach chosen by Mouradian et al. [33] is to classify the architectures depending on whether they are application-specific or not. They also elaborate on architectural challenges according to six criteria including scalability and heterogeneity. Naha et al. [34] describe a high-level architecture, works proposing different architectures, and propose a classification of fog components into eight functionality layers. Recently, Henze et al. [35] mathematically defined four concepts for describing fog architectures that account for the dynamic nature of edge environments.

### 2.1.3 Architecture Categories

In this thesis, the different architectures are grouped into three main categories inspired by the work of Mtibaa et al. [36]. Those categories are technology-independent and aim at visualizing three high-level variants of

---

1. [http://openedgecomputing.org/](http://openedgecomputing.org/)
2. [https://www.iiconsortium.org/index.htm](https://www.iiconsortium.org/index.htm)
3. [http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing](http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing)
The edge computing concept that the current works are using. The categories are used in Chapter 3 for classifying current works on resource management.

The first category, named *Edge server* and depicted in Figure 2.1, is a generic architecture where end devices are connected to an edge device of server type, which itself is connected to the rest of the network, including the cloud. In this type of architecture, the edge device is at a fixed physical location and has relatively high computational power, though it remains less powerful than a conventional data center used in the cloud computing paradigm. Moreover, there is a clear separation between the device level and the edge level. In the literature, such edge devices are named for example cloudlets [37, 38], micro data centers [39, 40], nano data centers [41] or local cloud [42]. They can be located for example in shops, enterprises, or co-located with the base stations of the telecom access network. Indeed, in the ongoing work on what the fifth generation (5G) of telecommunications networks will look like, a cloud radio access network (C-RAN) is envisaged [43, 44], with connections to other edge computing areas such as mobile cloud computing [45].

The second category, named *Coordinator device* and depicted in Figure 2.2, is an architecture where one end device acts as a coordinator between the other end devices. It also acts as a proxy towards an edge device and/or the
The difference between a coordinator device and an edge server is that the coordinator device can be mobile and has less computational power and bandwidth than an edge server. In this architecture category, the border between the device level and the edge level is not a sharp one, as the coordinator level providing edge functionality can actually be an end device. Solutions using this category of architecture are named for example fog colonies with a control node [46], vehicular clouds with a cluster head [47] and local clouds with a local resource coordinator [48].

The last category, named Device cloud and depicted in Figure 2.3, is an architecture where the end devices communicate with each other to find needed resources and deliver the wanted services. The devices might communicate with an edge device connected to the cloud if needed but this is not necessary. In this architecture category, the device level and the edge level are thus merged. Research work considering this category of architecture call it opportunistic computing [49], cooperation-based mobile cloud computing [50, 51], or transient clouds [52].
2.2 Mixed Reality

In this thesis, a case study is conducted on a specific edge application domain: mixed reality. Mixed reality is a very relevant use case for edge computing and is considered as a "killer app" for edge computing [27].

2.2.1 Reality-Virtuality Continuum

In a recent survey, Bekele et al. [53] present the reality-virtuality continuum and its evolution since the original description by Milgram and Kishino [54]. This continuum (shown on Figure 2.4) presents the span between reality on one side and virtual reality (VR) on the other side. In between are augmented reality (AR) and augmented virtuality (AV), depending on how much virtuality is embedded in the real-world.

Mixed Reality (MR) is the umbrella term for both AR and AV, designating any technology mixing reality and virtuality. Hence, an MR image can be including a few virtual elements, like one Android mascot in a room (see Figure 2.5a), or many virtual elements like real persons moving into a virtual city (see Figure 2.5b).
Figure 2.4: Reality-virtuality continuum.

Figure 2.5: MR examples.

(a) Few virtual elements  
(b) Many virtual elements
2.2. Mixed Reality

2.2.2 MR Pipeline

The basic steps of an MR application are illustrated in Figure 2.6. First the reality is captured, usually in the form of a video stream (1), then this stream is analyzed and a virtual representation of the world is constructed (called the point cloud) (2). After that, the virtual graphics are created and placed to be at the wanted position (3) and finally the resulting video stream is displayed to the user (4). Next, these steps are described in more detail.

Sensor input: The end device can be equipped with different sensors, the most common being a camera that will gather a video stream of the reality. This input can also be combined with other sensory input such as input from the Inertial Measurement Unit, to give more precise information about the end device’s movement in the surroundings.

Point cloud: In order for the virtual elements to blend nicely in the reality and therefore bring the illusion of reality to the end user, the MR application needs to create a virtual representation of the surroundings of the end device and to follow the location of the end device in this environment. For example, this representation will enable the detection of flat surfaces (horizontal or vertical) in order to place virtual elements at positions that make sense for a human (e.g. avoid placing a 3D virtual touristic guide so that it is walking on a wall).

In order to create this representation, simultaneous localization and mapping (SLAM) techniques have been developed in the past [55, 56]. Those techniques enable the creation of a 3D map of the surroundings and keep track of the end device’s position within it. This 3D map is often modelled as a 3D point cloud. This part is very computationally intensive since advanced computer vision are required (e.g. edge detection, human face detection, and structure from motion) to create the surroundings map.

Graphic overlay: Once the point cloud is created, it will be used to provide the MR functionality, for example to place the virtual element(s) when needed. A visualization of the point cloud can also be output to the user. Depending on how detailed the virtual objects should be rendered (only text or 3D objects with use of shadows, refined textures, etc.) the resource requirements for this part vary. After creating the virtual element(s), this part uses
the point cloud information to place each element at an appropriate location in the surroundings of the end device, thus creating an overlay to the initial video stream.

Display: Finally, the resulting video stream comprising the original video stream with the added virtual graphic overlay is displayed to the end user.

2.3 Video Streaming

In the MR case study conducted in this thesis, the sensor input (in this case a video) is sent to an edge device in order to be analyzed. To achieve this, video streaming techniques were required. These can be divided into two categories: the video compression standard that will be used for encoding/decoding the video and the transport protocol that will be used for the actual sending of the encoded video.

2.3.1 Video Compression Standards

In the thesis, two alternatives for the video compression standard are considered: H.264 and MJPEG. Their particularities are briefly presented here.

When using the H.264 standard, some video frames from the stream will be selected as keyframes. These are going to be transmitted entirely. The frames in between two keyframes are not transmitted entirely, only the difference to the previous frame is transmitted. The full image will then be reconstructed at the receiver end. This technique is used to reduce the amount of data that needs to be transmitted, at the expense of a higher complexity of the encoding/decoding phase.

In the MJPEG (or Motion JPEG) standard, every frame is compressed to a JPEG image. Depending on the degree of compression chosen at the encoding, the amount of data to be transmitted will be more or less important, at the expense of image quality.

2.3.2 Transport Protocols

As for video compression, two different alternatives are considered for the transport protocol used in the case study.

The first protocol considered is the Transmission Control Protocol (TCP). It is a protocol that was created for enabling reliable transmission for the Internet and was specified in the early 80s by the Internet Engineering Task Force. The focus of the TCP protocol is on ensuring that all the bytes transmitted will be received, in the same order as they were sent. This is achieved through an acknowledgment technique called positive acknowledgement with re-transmission. It means that the receiver should send an acknowledgment message back to the sender upon receiving a packet. If no acknowledgment is received within a timer, the sender will send the packet
2.3. Video Streaming

again. Thus, TCP offers no guarantee about how long it will take for a packet
to be delivered, but it will ensure that all packets will be delivered in the or-der they were sent.

The second protocol considered is the User Datagram Protocol (UDP). It was also designed in the early 80s by David P. Reed and belongs to the Inter-net protocol suite, like TCP. UDP is connectionless, meaning that it does not require some connection establishment mechanism. It also offers no guar-an tee about packet delivery or order. There will be no re-transmission if a packet gets lost. Therefore, the protocol is more lightweight than TCP, but it will be more sensitive to disturbances in the network.
In this chapter, an overview of resource management areas is presented. It includes the proposed taxonomy constructed through a literature review and presents state-of-the-art research in the field of resource management for edge computing. For the areas specifically discussed in more details in the rest of the thesis, related works that may consider other domains are also included when relevant.

3.1 Overview

As mentioned earlier, resource management is a crucial area for the successful deployment of edge computing. There is an increasing body of work in this broad area and the rest of this section gives an overview of the different perspectives that exist and presents the notion of offloading, that is used in various edge scenarios and exhibits challenges with regards to resource management.

3.1.1 Different Perspectives

In an early review, Yi et al. [57] include three works in a resource management section, writing that this is an interesting problem to be studied in the edge computing context.

Mukherjee et al. [58] include resource management issues as one aspect of their study, together with architectures, models used for latency, networking and energy consumption, issues in the access network and applications. However, they do not provide a taxonomy and only consider 12 works, mostly from the year 2016.
Mahmud et al. [59] include a resource-related branch in their taxonomy for fog computing. This one covers the resource or service provisioning metrics used by current works.

Yousefpour et al. [60] propose first a comparison of fog computing and related paradigms such as edge or cloud computing. Then, they present a taxonomy for fog computing that cover nine various aspects such as frameworks, design, software, hardware, and security. One of this aspect is resource management and provisioning, and the authors present part of the recent work in this area. Finally, Yousefpour et al. list nineteen challenges or future research directions, of which twelve are related to resource management, thus showing the importance that this area has for the future of edge computing.

Naha et al. [34] also propose first an overview of fog computing and of the related paradigms. Then they deep dive into the different architectures proposed. Similar to the architectures presented in Section 2.1.3, most work considered three layers. They propose a taxonomy for fog computing from the requirements perspective for the fog infrastructure, platforms and applications. Resource allocation and scheduling can be found as a sub category of the platform part.

Mach et al. [30] survey works dealing with computation offloading to mobile edge computing. They especially focus on presenting works dealing with the resource allocation aspect and on mobility management.

Mao et al. [29] consider the communication perspective on mobile edge computing and present different models used for computation and communication. With regards to resource management, they separate single-user systems from multi-user systems and classify works according to this. They also consider systems with heterogeneous servers.

Rejiba et al. [61] focus on one specific aspect of resource management and survey works dealing with service migration. Their classification is based on the type of objective chosen for the optimization of the migration performance: either it is about cost, time or success rate.

This chapter complements the above perspectives by providing an overview that is both i) specifically targeting resource management and not any other aspects of edge computing, and ii) covering all different aspects of resource management with a focus on edge resources. The notion of edge computing used for this review encompass the different architectural categories that were presented in Section 2.1.3. Figure 3.1 shows the distribution of the selected papers with regards to the three architectural categories. It can be seen that the edge server category is clearly predominant. This may be explained by a strong influence of the cloud computing community on the edge computing one, as the edge server category presents similarities with the way cloud computing is organized.
3.1. Overview

3.1.2 Offloading

One area that is of high importance and is present in many use-cases in edge computing is offloading. This is the idea of executing a task on another device than the current execution target, typically one having more powerful computational capacities or being less energy-constrained. Resource management is tightly connected to offloading since in order to take a decision to offload one needs to have knowledge about system resources. This knowledge is provided by resource management techniques. For example, resource discovery can be used as an input for taking an offloading decision while resource allocation techniques can be used to perform the offloading decision.

This generic idea of edge offloading is illustrated in Figure 3.2. The input can for example be sensor data such as a video stream or a GPS location, and the output something that will be displayed to the end users. The arrows depict communication, that can be internal to the device (in the non-offloaded case) or in between two devices (in the offloaded case).

Offloading to the cloud, that can be considered as having unlimited resources, is one of the solutions that has been envisioned in the past [62, 63, 64, 65, 66]. However, with the recent advent of edge computing, works such as the one by Chen et al. [27] have shown that using edge is clearly preferable to using cloud in the case of augmented reality applications, because of the reduced latency delay that edge provides.

In addition to latency considerations, offloading to the edge is also preferable from an energy point of view as the telecommunication networks have the highest share of the overall energy consumption of ICT. The fact that edge computing consists of local traffic is thus a step in the right direction [67]. Keeping the computation as close as possible to the end users would also keep the energy cost of edge applications as low as possible.
3. Resource Management at the Edge

Figure 3.2: Generic idea of edge offloading.

In the thesis, the case of an application that gathers input and outputs results in the same end device is considered (as shown in Figure 3.2). Other works like the one by Trinelli et al. [68] considered the case where the input and output devices are different.

3.2 Taxonomy

A taxonomy of resource management at the edge is proposed, that will be used for classifying the reviewed papers. This taxonomy, illustrated in Figure 3.3, aims at getting an overview of state-of-the-art research in the area of resource management in edge computing and presents four main aspects: resource type, objective of resource management, resource location, and resource use.

The two first aspects were constructed by reviewing the current type of resources used and the objective for which they are used in the literature. The two last aspects are based on mutually-exclusive pairs for describing the resource location and the use of the resource.

In the coming sections, the different parts of the taxonomy are described, as well as how the surveyed works can be placed with regards to the four above contexts.

3.3 Resource Type

When looking at resource management, the first step is to analyze what resource types are of interest in the particular context of edge computing. One reason to use edge is reducing the response time, which can be done if computation and communication resources are provided and used adequately. Storage as a resource is also a concern since local storage may benefit security or timeliness. A less obvious resource type is data (e.g. availability of sensors) that provides local benefits to an application (e.g. use of cameras
or location sensors). The next category, energy as a resource, is clearly influenced by the amount of computation, communication, storage, and data capturing that goes on. Finally, some works consider resources in a generic
way using abstract terms such as "Virtual Resource Value" [39, 69] or just as unit-less elements in a model.

### 3.3.1 Single Resource Focus

Even though the majority of the reviewed papers choose to focus on several resources, some works focus on only one resource type. In this case, most of the works use a generic resource type, that is used as an abstraction for actual resources. The abstraction used varies in various articles. For example, Penner et al. [52] work with device capabilities as an abstraction, and Wang et al. [70] use generic cost functions that can be used to model many aspects of performance such as monetary cost, service access latency, amount of processing resource consumption or a combination of these.

Moreover, some works focus solely on one resource type. Energy is one of them and Borylo et al. [71] classify datacenters in two categories (green and brown depending on which source of energy they use) before using a latency-aware policy to choose a data center for serving a request. Another one is computation. For example, Fricker et al. [72] use servers as an abstraction (one request occupies one server).

### 3.3.2 Multiple Resource Focus

The most common combination of resource types studied is computational and communicational resources together. Computational resources can be be addressed at a physical level, e.g. discussing CPU cycles [45], Millions of Instructions per Second (MIPS) [40], or number of processors per cloudlet [73]. It can also be addressed at a conceptual level by considering different computing resources based on the average number of tasks completed per unit of time [74], or by the use of virtual machines (VMs) [75] or containers [76] as resource elements. For communicational resources, bandwidth [77, 78], communication power [79], or a delay term impacting the task completion time [74, 40] are considered.

Some works include another resource type in addition to computation and communication. Habak et al [20] also consider data about task dependencies. Skarlat et al. [46] add service demand with regards to storage resources. VMs can also be considered as an encapsulation of computation, communication, and storage, in methods that ensure that underlying resources in the device hosting the VM are adequate [80]. Chen and Xu [81] consider computing (CPU cycles per second), communication as radio-access provisioning, and energy used both for transmission and computation. Sardellitti et al. [82] aim at minimizing the overall energy consumption of the users in their joint optimization problem. Finally, Elgazar et al. [83] propose a solution for file storage at the edge that consider the energy con-
sumption of the device involved as well as the communication technology used.

3.3.3 Summary of Resource Types

Examining the collection of papers considered, resource studies so far seem to focus on computation and communication resources to a greater extent. Moreover, data and energy are underrepresented among resources studied.

It is noticeable that storage is not the main focus of attention. This could be due to the fact that the cloud is available as a fall-back in many cases or that persistent data storage is not the main focus of most edge applications. An alternative explanation could be that the authors focus on a reduced set of resources for ease of presentation thinking that the work can be extended to other resources. Such claims, however, have to be considered with care as this is ignoring that interactions between resources may exist as studied by Yousaf et al. [44].

Some resources are dealt with mainly as physical elements whereas others naturally lend themselves to be defined in abstract ways. For example, sensors are present in the end devices, which can produce useful data needed for the completion of the task (as in [52, 48, 84, 46]), whereas bandwidth (throughput) is a natural abstraction for distinguishing between different radio interfaces or different physical environments. Moreover, when using a generic resource representation, it is easier to combine several resource types or to combine resources with other performance-related considerations, one example being the generic cost function in the work by Wang et al. [70].

3.4 Objective

An important aspect of resource management included in the taxonomy is its objective, that can be decomposed into several areas addressing different problems. This is shown in the branches under objective in Figure 3.3. Table 3.1 presents which problem(s) are addressed by a selection of articles.

By far, the most active area of research in the edge resource management is resource allocation, as visible in Table 3.1. This is followed by optimization as a goal, where we see a great majority of papers present. Among the objectives from our taxonomy, resource sharing and resource discovery are least studied.

3.4.1 Resource Estimation

One of the first requirements in resource management is the ability to estimate how many resources will be needed to complete a task or to carry a load. This is important, especially for being able to handle fluctuations in resource demand while maintaining a good quality of service (QoS) for the
## 3. Resource Management at the Edge

Table 3.1: Selected articles according to objective of resource management.

| Objective | Estimation | Discovery | Allocation | Sharing | Optimization |
|-----------|------------|-----------|------------|---------|--------------|
| Ahvar [67] | ✓          |           |            |         |              |
| Aït-Salaht [85] |             | ✓         |            |         |              |
| Bittencourt [78] |           | ✓         |            |         |              |
| Borgia [86] | ✓          |           |            |         |              |
| Borylo [71] | ✓          |           |            |         |              |
| Boubin [87] | ✓          |           |            |         |              |
| Chen [81]   |            | ✓         |            |         |              |
| De Maio [88] |           | ✓         |            |         | ✓            |
| Elgazar [83] |           | ✓         |            |         |              |
| Fahs [89]   | ✓          |           |            |         |              |
| Fricker [72] |           | ✓         |            |         |              |
| Gu [80]     | ✓          |           |            |         |              |
| Habak [20]  | ✓          | ✓         |            |         | ✓            |
| Liu [77]    | ✓          |           | ✓          |         |              |
| Liu [50]    | ✓          | ✓         |            |         | ✓            |
| Mahmud [90] | ✓          |           | ✓          |         |              |
| Penner [52] | ✓          |           |            |         |              |
| Rodrigues [73] |       | ✓         |            |         |              |
| Sardellitti [82] |      | ✓         |            |         |              |
| Singh [40]  | ✓          |           |            |         |              |
| Skarlat [46] |           | ✓         |            |         | ✓            |
| Sonmez [91] | ✓          |           | ✓          |         |              |
| Tang [79]   | ✓          |           |            |         |              |
| Tärneberg [75] |       | ✓         |            |         | ✓            |
| Wang [45]   | ✓          |           | ✓          |         |              |
| Wang [70]   | ✓          |           | ✓          |         | ✓            |
| Wang [92]   | ✓          |           | ✓          |         |              |
| Wang [93]   | ✓          |           | ✓          |         |              |
| Xia [94]    | ✓          |           | ✓          |         |              |
| Yi [76]     | ✓          |           | ✓          |         |              |
| Zamani [74] | ✓          | ✓         |            |         | ✓            |
user. On the supply side, resources at the edge can be mobile, and thus become unreachable, which makes them less reliable than resources in a data center. On the demand side, user mobility implies that there can be sudden user churn, with the corresponding dynamic requests having to be handled by the edge.

In their work, Liu et al. [50] use the average of historical data in order to predict the characteristics of resource distribution and usage for the next time slot. Wang et al. [70] study the optimal size of a look-ahead window for prediction into the future in order to minimize cost over time. With respect to computational resources, Habak et al. [20] are estimating the task requirements within a job analyzer.

Focusing on energy, Ahvar et al. [67] present a model for estimating the energy consumption of different edge and cloud architectures. Similarly, Boubin et al. [87] describe a model for estimating how a certain hardware/software configuration will impact mission throughput.

### 3.4.2 Resource Discovery

As opposed to the estimation problem which relates to the demand side, resource discovery is about the supply side. A management system needs to know which resources are available for use, where they are located and how long they will be available for use (especially if the resource providing device is moving or is battery-driven). This area is especially important at the edge where every resource is not under the control of the system at all times, so the supply is volatile.

Using a locally-centralized strategy for resource discovery as in [47] comes at the cost of the necessity to regularly update the node gathering the resource information. Such updates are costly, for example in terms of energy consumption, as studied by Liu et al. [50]. They propose an algorithm enabling a switch between a locally-centralized mode and a distributed mode. Some works include resource discovery through existing tools: Zamani et al. [74] use a framework called CometCloud which performs resource discovery for video analysis.

### 3.4.3 Resource Allocation

Resource allocation can be tackled from two different perspectives: where to allocate (both initially, but also where and when to perform a migration if needed), and when and how much to allocate. Among the dominant approaches to allocation in the selected articles presented in Table 3.1, we find the following three perspectives: placement (15 articles), migration (3 articles), and scheduling (2 articles). Some works also use a multi-perspective approach (6 articles).
3. RESOURCE MANAGEMENT AT THE EDGE

Placement
Most of the surveyed works emphasize task placement, i.e. where should the task be executed and the resource allocated for the best possible execution. The definition of best execution varies depending on the considered system and the focus of the research. The reviewed works consider placement for offloading from the end devices to the edge or the cloud \[82, 46, 70, 86, 83, 94\], or in between edge devices, i.e. for load distribution [72]. Some works consider dynamic resource provisioning [71], VM placement [80], or container placement [89].

As task placement is one part of the orchestration framework presented in the thesis, a further literature study showed that in the majority of the works, a central entity gets all the task placement requests. In this type of scenario, the task placement problem can be formulated as an optimization problem where several tasks have to be dispatched among several edge devices and is solved using various optimization techniques \[95, 70, 96, 97, 85\]. Even when the requests are received in a distributed manner among groups of edge devices, the task placement is still performed at only one device within the group [98]. Some works proposed heuristics instead of optimization [99, 94], but the task placement is still performed in a centralized manner. Foggy [100] is one example framework for centralized orchestration. However, centralized approaches have drawbacks and may not be suited to the inherently distributive nature of edge computing. Therefore, it is interesting to consider fully distributed task placement, i.e. where each edge device is responsible for placing the tasks it receives.

Distributed task placement problems are typically tackled by a pool of edge devices when the needed resources are not present at the current edge device. Guiguis et al. [101] present a distributed way of maintaining the resource information, when optimizing the placement of a set of tasks. Mascitti et al. [102] find the best composition of services between different devices. However, not only the tasks need be placed, but also the edge devices that will provide the resources for handling the tasks. In the selected works, only De Maio and Brandic [88] consider this aspect, but for placing stationary edge devices. This is also the case in the work by Li et al. [103]. Mobile edge devices have, to the best of our knowledge not been considered for edge servers, but Plachy et al. [104] showed that mobile (in that particular case flying) base stations are beneficial for increasing the data rate a user can get.

Migration
Still considering where the task should be executed and when it comes to virtual entities such as services, applications, tasks, and VMs, the focus could also be on how they can be moved during execution if the new location is better, i.e. on migration. For example, Tärneberg et al. [75] study application-
driven placement and Penner et al. [52] concentrate on task assignment that can be balanced among devices. With respect to virtual computational resources, Fan et al. [105] and Rodrigues et al. [73] focus on VM migration, but with different optimization objectives (increase the use of green energy and minimize delays, respectively).

**Scheduling**

Regarding when to allocate resources, Bittencourt et al. [78] study the impact of three different fog scheduling strategies on application QoS (concurrent, first come-first served, and delay-priority). With the same focus, Singh et al. [40] consider scheduling only for private tasks, i.e. that can only be executed on the local edge server. In their algorithm, tasks are considered in an earliest-deadline-first manner. Regarding how many resources to allocate, Wang et al. [45] propose a joint cost-effective resource allocation between the mobile cloud computing infrastructures and the cloud radio access network infrastructure.

**Multi-perspective**

Some works tackle several of the above mentioned perspectives. Liu et al. [77] present a multi-resource allocation system which first decides whether the request should be served or rejected (admission control), then where to run it (edge or cloud level), and finally how much bandwidth and computing resources should be allocated for this task. In the context of video analysis, Zamani et al. [74] also studied scheduling and placement. Their scheduling is based on identified chunks of video, applying two alternatives: minimizing computation time or minimizing computation costs. Their placement is done after resource discovery using CometCloud. Also in the area of video analytics, Yi et al. [76] investigate three task prioritizing schemes (Johnson’s rule, short IO first, longest CPU last) for scheduling the task requests at a receiving edge node and three task placement schemes for collaboration within the edge level (shortest transmission time first, shortest queue length first, and shortest scheduling latency first). Singh et al. [40] consider both placement and scheduling with respect to semi-private or public tasks. Still considering earliest-deadline first for scheduling, the placement strategy is to try first one’s own edge, then one’s own cloud and if they are overloaded, go to some external edge, and then to an external cloud. Habak et al. [20] first consider placement for deciding in which end device a task will be run and then scheduling of the computation resource. They propose a fair queuing-based task pick-up that ensures a fair execution of the tasks belonging to different services. Wang et al. [92] consider placement of user workload with mobile user and include migration as one of the four types of costs included
3. RESOURCE MANAGEMENT AT THE EDGE

in the optimization problem, together with operation cost, service quality cost and reconfiguration cost.

Other works that consider several perspectives for resource allocation are the ones proposing orchestrations frameworks. A few of them have been developed for edge computing in the past few years. Santoro et al. [100] propose a framework called Foggy for workload association. They implemented the framework in a testbed by using the OpenStack and Kubernetes technologies and the orchestration is performed in a centralized way. Sonmez et al. [91] propose a workload orchestrator using fuzzy logic evaluated using the EdgeCloudSim simulator. The orchestration is performed in a centralized way and considers only task placement. In a recent licentiate thesis, Nguyen [106] describes a proposal for an autonomous edge resource management system using a MAPE-K (Monitor, Analyze, Plan, Execution, and Knowledge) feedback loop. The implementation of this system is a work-in-progress and so far the focus has been on studying workload prediction for the Analyze part of the system.

3.4.4 Resource Sharing

Resources on end devices are heterogeneous and most of the time scarce. Edge devices also have limited resources compared to (almost infinite) resources in the cloud. Sharing resources between devices or between end and edge devices aims at tackling three different issues: not having the needed resource at all in the device where the task is initiated, not having enough of it, or using other devices’ resources in order to get a faster completion of the task. Sharing resources is typically realized by pooling resources in the local vicinity of client nodes. This can extend to the edge domain (clustering edge servers) or remain at end devices. The latter is investigated by Skarlat et al in so-called fog colonies [46].

We can classify the reviewed articles into two categories according to whether they include how to form the groups of devices that will share resources or if they assume that the formation is already done and focus on the actual sharing. We call these two categories as dynamic coalitions, and static coalitions respectively. Starting with dynamic coalitions, Chen and Xu [81] include the formation of device coalitions by using a coalition game incorporating trust considerations. When supply matches demand, they found that using a coalition can lead up to 40% lower weighted cost compared to a non-cooperative scenario. Moving to static coalitions, Skarlat et al. [46] consider resources shared between two neighbor fog colonies and achieve a 35% reduction of execution cost compared to a cloud-only strategy.

Finally, even if resource sharing can bring benefits for a group of end devices, it is not obvious that users will agree to share their resources, especially if they are always on the providing side. Therefore there is a need to develop incentives for resource sharing. This is tackled in works by Tang et
al. [79] and Chen and Xu [81]. The following mechanisms are provided in the above works respectively: a) a double bidding mechanism for demander and supplier of resources where the focus is on how to encourage mobiles with resources to share them, and b) payment incentives for lending out resources. On the same topic, Habak et al. [20] performed a pilot study to identify effective incentive mechanisms such as getting compensation (e.g. money) or if the reason for the computation taking place is significant (e.g. emergencies).

3.4.5 Resource Optimization

A fifth objective pursued in the reviewed works is to optimize the resource use at the edge. This is usually a joint objective together with one of the previously described objectives. Which aspect should be optimized and the associated constraints varies among the reviewed works but the three main ones are QoS (often understood as latency), energy, and operational cost. How the optimization problem is formulated and solved also varies, and those variations are presented below.

First, some articles consider selecting the optimum solution by comparing the results from different candidates and selecting the minimum/maximum value depending on the objective [44, 107]. Another group of works solves their optimization problem using linear programming [77, 41], an approximation based on linear programming [51], or integer linear programming [46, 108]. Other works use mixed-integer non-linear programming [80, 76]. Recently, constraint programming [85] has also been used.

Some works focus on convex problems [70, 48], while other consider non-convex problems and either tranform it to a convex problem (e.g. using a Weighted Minimum Mean Square Error-based method [45]) or solve it (e.g. by developing a method based on Successive Convex Approximation, for the centralized approach in [82]).

A further group of works proposes their own algorithm or heuristic. Tärneberg et al. [75] approximate an exhaustive search approach yielding an optimal solution but having exponential computation complexity with an iterative local search algorithm finding a local optimal solution. Zamani et al. [74] implement an optimization strategy where constraints on computation time and cost are enforced using an admission control strategy. Wang et al. [70] present a binary search algorithm for finding the optimal lookahead window size, and Habak et al. [20] propose an algorithm in order to do deadline-based optimization when a helper has to handle multiple tasks belonging to different services. Liu et al. [50] propose a heuristic algorithm that uses different statistics to estimate the energy that is going to be consumed in each of the two possible modes during a time slot, and chooses which mode to use depending on this and other parameters. Mahmud et al.
[90] propose an heuristic for reducing the number of active edge nodes, as a linear programming solution takes too long to be found. Xia et al. [94] propose four different heuristics for minimizing the average response time that can be combined to each other in order to improve scalability and response times. Finally, De Maio and Brandic [88] use the meta heuristic NSGA-II, which is a genetic algorithm.

A recent body of works consider online methods for optimizing. Indeed, the arrival times or any other required task information may not be available in advance. For their online solution, Sonmez et al. [91] use fuzzy logic for their workload orchestrator while Wang et al. [92] propose the MOERA algorithm for solving their formulated mixed nonlinear problem using the regularization technique and convex programming. Also considering online placement, Wang et al. [93] propose an approximation algorithm building upon a baseline algorithm giving an optimal solution in the case of a linear application graph.

3.5 Resource Location

Computing at the edge differentiates itself from regular cloud computing with the fact that resources used can belong to different levels. It is indeed not uncommon to use resources at the edge level primarily, but also from the cloud level if required. Moreover, end devices, and sometimes edge devices do not have to be stationary as in a data center. Note that here we make a distinction between mobility on the demand side and mobility on the supply side. Even though the demand side clients are almost always mobile, the infrastructure that supplies the adequate resources has been invariably stationary in the past.

3.5.1 Location within the Architecture

As expected, when studying the selected edge resource management papers a large part (35%) of those presented in Table 3.1 consider managed resources located only at the edge level, for example, the works by Chen and Xu [81], and Sardellitti et al. [82]. Gu et al. [80] consider resources on different base stations, while Fricker et al. [72] and Rodrigues et al. [73] consider different types of edge devices. Essentially refining the architecture, some works distinguish different levels between the same architectural level. For example, Tärneberg et al. [75] consider that data centers at the edge can have a different distance to the device, and different sizes. Sometimes, works consider resources located only at the device level but where the management is performed at the edge [79].

However, the resources managed do not need to belong to the same architecture level. Among the multi-levels works, the most common is to use resources located both at the edge and at the cloud level [77, 71, 76, 70, 40, 50].
3.6 Resource Use

Specifically, Skarlat et al. [46] and Bittencourt et al. [78] favor using edge resources over cloud resources. This is, however, not the only combination and Habak et al. [20] work with resources located both in the end devices and at the edge since an end device is promoted to an edge role. Finally, combining the three levels, Zamani et al. [74] use resources on the device, on the network path to the cloud (edge level) and in the cloud level.

3.5.2 Resource Mobility

In an edge context, it is not obvious that resources located in the lower two levels of the architecture will be stationary or mobile.

In most of the selected articles for Table 3.1 (81%), the resources considered are stationary only. This can be because the architecture/application considered does not have mobile resources or for simplification reasons. The latter is found in works where the architecture presented has resources that are theoretically mobile but where this part is ignored in the solution or evaluation presented, e.g. in [50]. This preponderance of stationary resources may be explained by the fact that those works consider edge as an extension of the cloud, which has only stationary resources.

Note that this does not mean that the works do not consider mobility at all, only that the mobility is not on the supply side. Works including mobility on the demand side only are, for example, Plachy et al. [109] who consider that computational resources needed by a user are allocated in a stationary base station in a VM, which can be transferred to another base station if the user is moving. Similar solutions are presented by Tärneberg et al. [75] and Wang et al. [70].

Having mobile edge devices, and thus mobile resources obviously creates lots of challenges such as how to handle the unreliable connectivity of those resources, how to provide seamless handovers, etc. Thus, having mobile resources introduce another level of complexity in resource management algorithms. Different mobility models are used, for example, Penner et al. [52] model departure and arrival times using statistical models. Also using statistical models, Habak et al. [20] model arrival rate and presence time. In those statistical models, arrivals are modeled using a Poisson distribution, departure most often using an exponential distribution and presence time using a normal distribution.

Some few works mention a combination of mobile and stationary resources. For example, Borgia et al. [86] consider the local cloud (i.e. the edge level) as mobile and the global cloud as stationary.

3.6 Resource Use

The final aspect of resource management considered in our taxonomy is the purpose for which the resource will be used.
3. RESOURCE MANAGEMENT AT THE EDGE

3.6.1 Functional Properties

Edge computing is promoted as a means of getting access to a given service in most of the surveyed articles, i.e. for satisfying functionality in an application.

Edge Applications

There are numerous articles in the literature providing an overview of edge applications, including Liu et al. [10] that propose an edge application classification. Such applications range over smart agriculture [23], augmented reality [29, 25, 22, 34], disaster recovery [110], connected vehicles [29, 25, 6, 22], smart city [23, 34], healthcare [22, 34], and several others (e.g. [21, 5, 30]). However, those works do not study further the presented applications.

With the long term goal of creating a benchmark for edge computing algorithms and techniques, the SPEC group has been recently looking deeper into different use cases (mixed reality, vehicular data analytics, and code offloading) in order to characterize their workloads [111]. The thesis work with mixed reality is a part of this effort to look deeper at specific edge applications.

When looking at the different applications used in the reviewed articles presented in Sections 3.3 to 3.5 and in our survey article [112], the first finding is that the majority of them do not consider a specific application in their study. Instead, they refer to generic applications such as IoT services [113], real-time applications [109], latency-sensitive applications [77], or name some applications but only as an illustration.

Table 3.2 presents the remaining papers according to which type of application they consider. We can distinguish seven areas in which the described applications can be categorized. Note that in the Generic category we place papers that although not fixed towards one domain of application refer specifically to classes of applications that they exemplify clearly.

Modeling Edge Applications

When looking at the reviewed articles, a large part of the research in edge resource management use simulation or analytical tools, which require a model or a trace of the application considered instead of an implementation of this application. Therefore, resource management studies are performed with different application workloads. With regards to edge applications, the currently used models are most often not based on real applications but on characteristics that are derived from how the authors think such an application will behave. As the thesis is tackling edge application modeling, an overview of related works in this area is presented here.

For example, in a paper introducing the EdgeCloudSim simulator [116], Sonmez et al. present three applications (including an MR application).
Table 3.2: Applications considered in reviewed articles.

| Area            | Applications                                      | Articles |
|-----------------|---------------------------------------------------|----------|
| Healthcare      | Medical cyber-physical systems                    | [80]     |
|                 | Connected health                                  | [84]     |
| Video           | Video analytics                                   | [74, 76] |
|                 | Video surveillance                                | [78, 114, 84] |
|                 | Video on Demand                                   | [41]     |
|                 | Face recognition                                  | [85]     |
|                 | Agricultural scouting                             | [87]     |
| IoT             | Crowd-sensing                                     | [115]    |
|                 | Sense-Process-Actuate application                 | [46]     |
|                 | Smart bell                                        | [85]     |
| Gaming          | Electroencephalography (EEG) tractor beam game    | [78]     |
| Transportation  | Connected vehicles                                | [42, 47] |
|                 | Data Stream Processing for traffic knowledge      | [94]     |
| Content         | User profiling                                    | [13]     |
| management      |                                                   |          |
| Generic         | Computation/Communication intensive               | [73]     |
|                 | Delay-sensitive/Delay-tolerant                    | [108]    |
|                 | Event-driven                                      | [90]     |

Those are mostly described in broad terms such as "high/small" for the CPU resources required and the amount of data transferred. Apart from the upload/download data size for the MR application that are derived based on a chosen image/text metadata size, the other numerical data is not justified. Similarly, applications considered for the benchmarking effort by McChesney et al. [117] and Toczé et al. [111], have their characteristics described in broad terms such as "bandwidth/computational intensive" or "close/far".

Mukherjee et al. [58] summarize different mathematical models used for capturing e.g. latency in edge computing in earlier works [118, 119]. Such models are useful but they also have limitations. Since they are generic they do not capture the specificities of different application types, e.g. they include assumptions such as task arrivals following a Poisson process, and they do not provide numerical values based on empirical measurements.

Characterization and modeling studies have been performed for workloads in other domains, for example in the related area of cloud computing.

Shen et al. [120] performed a statistical characterization of business-critical workloads for cloud computing. They used three different statistical
3. Resource Management at the Edge

instruments, namely basic statistics, correlations and time-pattern analysis. They found interesting findings about how such a workload behaves with the aim of improving the resource usage effectiveness and handling peak loads in clouds.

Talluri et al. [121] propose a characterization of real workload traces from Big Data applications in a cloud computing context. They perform statistical analysis of the data and study different aspects such as long-term trends, impact of file types, and clustering.

3.6.2 Non-functional Properties

In addition to enabling functionalities when using the edge computing paradigm, there is a need to provide some kind of resource management for the organization of the edge architecture itself and the realization of desirable properties. This additional work is not directly related to the service to obtain, i.e. it is a non-functional property (also referred to as extra-functional property). Obviously, papers that are focusing on a functional property can also be interested in some non-functional property.

This subsection is related to the categories of objectives for resource management already discussed in Section 3.4. Achieving the objectives in that section was evaluated using metrics that are often representative for measuring non-functional properties.

Examples of metrics and their related non-functional property that are often encountered are:

- **response time** as a measure of timeliness.
- **energy consumption** as a measure of energy efficiency.
- **admission ratio**, or its equivalent blocking probability, as a measure of availability of the edge service.
- **CPU/network utilization** as a measure of computation/communication resource efficiency.
- **Monetary cost** paid to an infrastructure owner as a measure of cost efficiency.

The above list of metrics is not exhaustive, but comprises the most prevalent ones.

3.6.3 Application Focus: Mixed Reality

This section introduces related works for the application type considered in the thesis. It includes works outside of the edge computing area, but that are of relevance for the thesis.
Offloading for Mixed Reality

In a recent survey, Chatzopoulos et al. [24] describe the current state of the mobile augmented reality area and the need for offloading some of the heavy computations that are needed by AR applications. Such an offloading can be done on a companion device, in the cloud or at the edge. This offloading can be done in two ways: either the rendering is still performed locally or both the rendering and the computing are performed on the remote device. Another possibility is to use approximate computing but this technique is still at an early stage.

The idea of offloading MR tasks has not appeared with the edge computing paradigm and has been for example studied in the context of cloud computing. Luo [63] implemented a gesture interface running on a local cloud located within the University of Illinois that is rendered on a display wall. Naqvi et al. [64] implemented an enhanced indoor positioning system that can give the user extra information about where she is by analyzing the environment. They successfully offloaded the object recognition task to the cloud. However, the recognition latency was high, at best 2 seconds. Shea et al. [65] created an application similar to the Pokemon Go game where they offload part of the computation to the cloud. The rendering is performed in the end device. Huang et al. [66] propose a framework called CloudRidAR for creating AR applications using the cloud for offloading parts of the AR tasks. In their prototype, the rendering is done on the end device and the feature tracking at the edge as this part is more computation-heavy.

Mixed Reality at the Edge

As edge computing aims at reducing application latency, it is promising for MR and a few works have started investigating it. For example, Ahn et al. [122] study the filtering of AR application content using reinforcement learning techniques. They evaluate their proposed optimized policy in a custom-built AR simulator based on the Unity game engine. The same group made a later study [123] about imitation learning for personalized AR, still using their custom AR simulator. Zhang et al. [124] study the use of several edge devices for performing the tasks of encoding and rendering, in order to improve quality of service. They evaluated their optimized solution using a test application.

In their work, Chen et al. [27] study prototypes for seven different wearable cognitive assistance applications, that are thought to be running on smart glasses. They use their Gabriel platform [125] for implementation. Their evaluation study includes a comparison of running the application at the edge or in the cloud, using 4G or WiFi for the first hop connection, and a comparison of using smartglasses and mobile phones.
3. **Resource Management at the Edge**

Recently, Trinelli et al. [68] investigated Network Function Virtualization for providing computing acceleration for MR by offloading to the edge. The MR application type considered was object detection using machine learning techniques. In their framework, called NEAR, they consider that the device creating the video stream is different from the one displaying the MR-enhanced stream.

Object recognition using machine learning is also the focus of Zhang et al. [126]. Their system called Jaguar can recognize an object with low latency, after an offline training phase of the edge part. In their work, some preprocessing of the video frames is performed on the end device.

The field of MR and especially AR is very active and one of the latest realization platforms is WebAR. In WebAR, a browser is used to provide an AR solution that is supposed to be lightweight and cross-platform. Qiao et al. [127] evaluate the performance of a WebAR implementation that showed lower latency when run at the edge compared to when running on the end device. However, application MR is still the most commonly used scenario while WebAR is in the emerging phase [127].

**Communication Aspect of Mixed Reality**

There are only few studies comparing different transmission protocols in client-server relationships in MR-related contexts. For example, Fernandez et al. [128] found that UDP allows twice as many supported clients as TCP in a collaborative augmented reality scenario. However, in this case, the messages exchanged between the clients are about the location, and not about video streams.

When implementing their Gabriel prototype, Ha et al. [125] tested both TCP and UDP and concluded that the difference between them in terms of response time was negligible when transmitting over Wi-Fi. They then decided to use TCP to avoid the complexity added by UDP for maintaining reliable transmission.
In this chapter, the focus is on the resource supply aspect and a study of the impact of introducing dedicated mobile edge devices in addition to the traditionally-used stationary ones is presented. A distributed framework (named ORCH) for managing the edge resources is proposed in order to provide high QoS under temporally and spatially changing load.

4.1 D2TEP Problem

When adding mobile edge devices to stationary ones on the resource supply side, a placement problem for the mobile edge devices is added to the load placement problem. I.e. in addition to having to decide where to send a task for execution, the edge system also have to decide where to send the mobile edge devices. This two-fold problem is referred to as the Distributed Dynamic Task and Edge Placement (D2TEP) problem, that is composed of a task placement subproblem and a mobile edge placement subproblem.

The task placement subproblem can be formulated as follows: for each task \( r \) incoming to the closest edge device \( e \), \( e \) has to decide on which edge device \( e' \) the task \( r \) will be allocated so that \( e' \) has enough resources to compute \( r \) before its deadline.

The mobile edge placement subproblem can be formulated as follows: at every load change, a location for all the mobile edge devices should be found such that the incoming load is served with as high success ratio as possible.

The D2TEP problem is distributed since task placement can be done by all the edge devices (and not only by a centralized placing system), and it is dynamic as the mobile edge placement is dynamically updated depending on the changes in the load volume and locality over time.
4. Distributed Orchestration Using Mobile Edge Devices

4.2 Considered Scenario

The scenario used is the following: in a smart city, edge infrastructure providers have to handle a load coming from mobile end devices. The edge infrastructure providers own and control edge devices for being able to serve this load: this is the resource supply part.

We consider that the load consists of tasks with various requirements coming from end devices with mobility over various locations in the city space. Hence, the actual load will depend on where the different end users are located and what they are currently requiring for service, i.e. which type of task they will generate. Moreover, we consider that the computation required for a task can be performed by any edge device. The challenging part of the scenario is that some tasks have very strict real-time requirements that have to be satisfied to provide the service they are part of. As the time spent transmitting to the edge has a significant impact on the total task latency, we consider that tasks with stringent real-time requirements have to be served in the vicinity of the end devices issuing them.

4.3 Models

In order to study the complex scenario presented in Section 4.2, we model its different parts and present the resulting models in the rest of this section.

4.3.1 System Model

We start by the model used for describing the devices involved in the scenario, i.e. the system model. We use a three-layer system model based on the coordinator device architecture described in Chapter 2. This model is presented in Figure 4.1. There are three types of devices involved, one type per layer:

- Cloud device (stationary, black cloud)
- Edge device (mobile or stationary, gray rectangles)
- End device (mobile or stationary, gray triangles)

A dashed outline in Figure 4.1 indicates a mobile device.

In addition to this three-layer division, we introduce the notion of edge orchestration area to represent the spatial distribution of end devices connecting to the system and to reflect the way the generated load will reach the edge devices in that space. The cloud services are common for the edge orchestration areas but each end or edge device only belongs to one edge orchestration area. In the rest of the chapter, we will focus on only one edge orchestration area, but the work can trivially be scaled up to several areas. It may also be
extended to overlapping areas with some notion of handover, but we begin with the simplest instance.

Each edge orchestration area is composed of $M$ end devices and a set of $E$ edge devices $\mathcal{E} = \{e_1, ..., e_E\}$. A subset of $\mathcal{E}$ denoted as $\mathcal{E}^{\text{mob}}$ contains the mobile edge devices. At any point in time, one edge device $e \in \mathcal{E}$ acts as the edge device responsible for the area orchestration in each area. This edge device is referred to as the area orchestrator in the ORCH framework. It is depicted as a rectangle with a conductor icon in Figure 4.1 and will be denoted $o$ in the rest of the chapter. The cloud device used when resources outside the edge orchestration area are needed is denoted $c$.

Within an edge orchestration area, each edge device $e$ considers a subset $\mathcal{N}$ of $\mathcal{E}$ as edge devices that can be used for offloading a part of the load coming to $e$. Those edge devices are called neighboring devices.

### 4.3.2 Area Model

Since the locality of the devices plays an important role in our scenario, there is a need for modelling the geographical area considered and the locality of devices within it.

Our area model considers that edge devices are only able to serve requests when located at specific positions called serving positions, and not while moving between those. For example, the edge device could be a drone that has two modes: either its resources (e.g. energy and computation) are used for serving edge requests at a serving position, or its resources are used for navigating between two of those serving positions.
Moreover, in our scenario, the exact location of an end device is not required as long as it is in the connectivity range of some edge device. Therefore, the geographical area is represented by $S$ segments, where $S = \{s_1, ..., s_S\}$ and where the physical area included within segment $s_i$ corresponds to the connectivity range of an edge device $e$ located at the serving position $\rho_i$ within $s_i$. The set of adjacent segments to segment $i$ is denoted as $A_i$.

Figure 4.2 shows an example of how a real neighborhood at Vallastaden in Linköping could be divided according to this model. The serving positions are denoted as yellow circles and the segments are delimited in black. In a real environment, the shape of segments will depend on the hardware used for communication and how the physical environment looks like. Details on how this is done are out of the scope of this study, where we assume that the segments are already defined.

Each edge device $e$ is associated with the segments that it can serve. Those segments are denoted as the set $S_e \subseteq S$. A mobile edge device can only serve the segment that it is currently located in (hence for $e \in E^{\text{mob}}$, $|S_e| = 1$), whereas stationary edge devices can serve several segments. It is assumed that 1) all end and mobile edge devices know their current segment, and 2) an end device $m$ located in segment $s_m$ has a mechanism to establish communication with the closest edge device. The closest edge device is defined as a mobile edge device located within the same segment, and if none, a stationary edge device serving this segment.

### 4.3.3 Resource Models

In this study, we consider both computation and communication resources. As seen in Section 3.3.2, this is the most common combination studied, showing the importance of those two resources.
4.3. Models

Computation

Computational resources are considered at the edge and cloud level. Cloud level resources are assumed unlimited but edge resources are limited, for example, to a fixed number of Millions of Instructions Per Second (MIPS). The duration of the computation at a device thus depends on the size of the task to execute and the resources available at the device.

Communication

Regarding communication resources shared among all the devices, we assume unconstrained bandwidth in the system, meaning that communication time is equal to the transmission time between two devices. The model can later be extended by adding bandwidth constraints and queuing time.

The transmission time or delay $d_L$ for a task to move between devices depends on the type of the link $L$ and is defined as follows for the different links considered. Equation 4.1 shows the delay for a link $L$ that connects an end device $m$ in segment $s$ to an edge device $e$.

$$d_L = \begin{cases} 
    d_{\text{short}} & \text{if } s^m \in S_e \text{ and } |S_e| = 1 \\
    d_{\text{medium}} & \text{if } s^m \in S_e \text{ and } |S_e| > 1 \\
    \infty & \text{otherwise}
\end{cases} \quad (4.1)$$

The infinite delay models the case where an end device is outside of the coverage areas for the edge devices.

The delay between two edge devices is assumed to be similar to what it takes for sending a task from an end device to a stationary edge device, i.e. $d_{\text{medium}}$.

Finally, the delay for a link between an edge device $e$ and the cloud device $c$ is as follows:

$$d_L = \begin{cases} 
    d_{\text{medium}} + d_{\text{long}} & \text{if } |S_e| = 1 \\
    d_{\text{long}} & \text{otherwise}
\end{cases} \quad (4.2)$$

In an orchestration area, $d_{\text{short}}$, $d_{\text{medium}}$, and $d_{\text{long}}$ are configurable parameters.

Edge devices can communicate with each others to exchange relevant information (corresponding to signaling). This is assumed to have a negligible impact on the task communication times and is therefore not included in the above equations.

4.3.4 Load Model

The next part of the scenario that needs to be modelled is the load coming from the end devices. The load model includes a task model and an application model.
4. Distributed Orchestration Using Mobile Edge Devices

Task Model

Each task $r$ is independent and is characterized by (a) a computation requirement $R_r$ in Millions of Instructions (MI) and (b) a deadline $D_r$ in milliseconds. The characterization of tasks can be extended to include other resource requirements. The generation of tasks is modeled as a Poisson process.

In this model, the tasks fall into three categories, depending on their deadline requirements. Those categories are derived from the traffic classes presented by Bianzino et al. [108]:

- delay-sensitive (DS) tasks
- delay-constrained (DC) tasks
- delay-tolerant (DT) tasks

Tasks falling into the DS category need to be executed on an edge device located in the same segment as the end device in order to avoid communication link delays. In our study, this means it needs to execute on a mobile edge device or on a stationary edge device dedicated to this segment. An example of applications that could produce DS tasks are augmented or virtual reality applications.

Tasks falling into the DC category need to be executed on some edge device, but not necessarily an edge device in this segment. Examples of edge applications that could generate DC tasks are edge analytics applications such as face recognition in video-based surveillance.

Tasks falling into the DT category can be executed on an edge or cloud device. An example of applications producing DT tasks are delay-tolerant gaming applications.

Application Model

The above task categories are used to model two application types used by end devices: urgent applications and regular ones. Urgent applications are modeled by a generated task mix with a large proportion of DS and DC tasks, and few DT tasks, whereas regular applications are modeled by a task mix with a larger portion of DC tasks, few DT and no DS tasks. Moreover, urgent applications have a lower mean for the Poisson process generating the tasks.

End device mobility and applications launched by users naturally change the locality of the load over time. In this work, we refer to a sudden surge in DS traffic at a given segment as a load change.

4.3.5 End Device Mobility Model

To model the mobility of end devices, we adapt the Weighted Waypoint model presented by Hsu et al. [129] to the area model presented in Section
4.3. Models

4.3.2. Indeed, end devices are only able to move to neighboring segments so their movements between segments cannot be considered fully random as in the widely used Random Waypoint model.

Each segment has its own stay duration distribution which represents how long the end devices will stay in that segment, depending on the characteristics of the segment. For example, a mobile end device will probably stay less time in a segment with only apartment buildings than in a segment with shops or tourist attractions. On entering a segment, an end device gets a value for stay duration from the segment’s distribution. This work considers three types of segments: residential, commercial, and outside segments but the list can be modified or extended in different scenarios. The stay duration in each segment is characterized as an exponential distribution with two different rates: a small one $\lambda^s$ for residential segments and a large one $\lambda^l$ for commercial and outside segments. End devices located in an outside segment do not generate tasks.

In the considered end device mobility model, end devices can only move to adjacent segments and the transition probability $P_{ij}$ from segment $i$ to an adjacent segment $j \in A_i$ is the same towards each adjacent segment, meaning it is calculated as the inverse of $|A_i|$. For end devices, the transition between two neighboring segments is instantaneous as the time for the end device to move close to the segment border is modeled as part of the stay duration.

### 4.3.6 Edge Device Mobility Model

Since this work also includes mobile edge devices in addition to stationary ones, the mobility of edge devices needed its own model. We choose to use a two-state model.

Mobile edge devices can be stable, when they are located at a serving position $\rho$ and accept tasks. They can also be moving between a service position $\rho$ and a service position $\rho'$, not necessarily a neighboring one. In this state, they are unavailable for the system, and are therefore not considered when determining the closest edge device or performing mobile edge placement since they do not accept new tasks while changing segments.

How long an edge device will stay in each of the two states will depend on the mobility strategy chosen. The strategies considered for this work are described in details in Section 4.5.3.

The current model does not support migration of tasks, meaning that once a task coming from an end device $m$ has been placed on an edge device $e$ for execution, all the execution will be performed at $e$. When the result is available, if $e$ and $m$ are not located in the same segment anymore because one of them moved, the task is considered as failed. Extending the model would need to borrow the concept of store-and-forward from delay-tolerant networking.
4. DISTRIBUTED ORCHESTRATION USING MOBILE EDGE DEVICES

4.4 ORCH Framework

The orchestration framework ORCH, shown in Figure 4.3, is proposed to address the D2TEP problem formulated in Section 4.1. We begin with an overview of ORCH and then describe each component of it in more detail.

4.4.1 Overview

The ORCH framework comprises two different orchestrations: the edge orchestration and the area orchestration.

Each edge device performs the edge orchestration, which addresses the task placement subproblem of D2TEP and is depicted in Figure 4.3 with blue boxes. The task characterizer identifies characteristics of the incoming tasks, and the capacity estimator estimates the available capacity of the neighboring edge devices. Both are used by the task placer, which decides on which edge device the task is going to be executed. Based on the received tasks, each edge device also stores load information to be used by the area orchestrator. The edge orchestration takes place each time a task (or a configurable batch of tasks to be considered together) is submitted to an edge device.

One edge device per orchestration area is designated as area orchestrator at relevant decision points. In addition to the edge orchestration, this edge device also performs the area orchestration in order to address the mobile edge placement subproblem of the D2TEP. This second orchestration is depicted in Figure 4.3 by green boxes. The area orchestration components are run asynchronously when load changes require it. The area orchestrator role can change according to various schemes, e.g., using a rotation among the edge devices, with higher priority given to the stationary ones (or the ones with

---

Figure 4.3: Overview of the ORCH framework.
more available resources). This rotation scheme will not be discussed further in the thesis but can be similar to existing schemes [108]. The two components of the area orchestrator are the load change detector, which observes the area load and detects load changes upon which the mobile edge placer decides where the mobile edge devices should be moved to, based on information gathered by the framework. After the mobile edge placement is performed, the edge devices move to their new locations (when needed). Details of how this movement is performed is outside the scope of this work.

The ORCH framework is designed and implemented in a modular way so that the detailed algorithms and functions used in each component can be modified easily (see Section 4.5.1). It is thus possible to address the D2TEP problem using refinements or variations to the schemes proposed in Sections 4.4.3 and 4.4.4.

4.4.2 Approach

In this work, a strict deadline-aware approach is used to address the D2TEP problem. This means that the task requirements described in Section 4.3.4 are strictly enforced and deadline misses lead to task abortion. For example, short deadlines for DS tasks mean that an edge device will be dedicated to serving them in the segment where a surge is detected. If the deadline requirements cannot be accommodated, because there is no available resource to fulfill the needs, the task is deemed failed and no attempt to execute it will be made.

4.4.3 Edge Orchestration

The edge orchestration proposal is presented in Algorithm 1. It has two main parts: the first one (lines 1-6) corresponds to the task characterizer and task placer components. A count of DS tasks received per segment is maintained. Other information, such as task placement success ratio, could be stored if needed by the area orchestrator. The second part (lines 7-10) corresponds to the capacity estimator component.

Task Characterizer

In the ORCH framework, the task characterizer’s main role is to add information to the incoming tasks. This information is used for both task placement and load change detection.

Hence, the task characterizer will associate a deadline category to the tasks as a means of preparing them before being considered for placement. For the DS category, this is done by considering whether $D_r < 2 \times d_{medium}$, in which case $r$ is tagged as DS. A similar reasoning with $d_{long}$ is used to tag a task as DC. The remaining tasks are tagged as DT.
4. DISTRIBUTED ORCHESTRATION USING MOBILE EDGE DEVICES

Algorithm 1: Edge orchestration

// Runs at each edge device $e$

parameter: $T_c$ // Cap. est. period

state : $C = \langle n, c \rangle$ // List of capacity estimation $c$

for each neighboring edge device $n$

$L = \langle s, \#DS \rangle$ // List of # of DS tasks for each segment $s$

1 upon receiving task $r$ from segment $s$ do

// Task characterizer

2 $tag \leftarrow$ Characterize($r$);

3 if $tag = DS$ then

4 Increment $\#DS$ for $s$

// Task placer

5 $device \leftarrow$ PlaceTask($r, tag, e$);

6 sendForExecution($r, device$)

7 every $T_c$ do // Capacity estimator

8 $N \leftarrow$ GetNeighboringEdgeDevices($e$);

9 foreach $n \in N$ do

10 Update $c$ for $n$ with GetAvailRes($n$);

The task characterizer’s second role is to keep track of the incoming load by storing relevant information that will be used by the load change detector. In order to avoid unnecessary use of memory, this information is deleted once fetched by the load change detector. In this work, the task characterizer stores the number of DS tasks incoming per segment.

Capacity Estimator

In ORCH, the capacity estimator informs the edge device receiving the task about the current status of resource usage at the neighboring edge devices. This information is used by the task placer when deciding on task placement.

Having a capacity estimator at each edge device reduces the need for extra communication between the edge devices in order to get the resource available on each of them. This is especially useful if a lot of neighbours are available.

In this first proposal, only computational resources are considered for capacity estimation. Those resources are aggregated in a pool and measured as units of available VM. The capacity estimator will periodically (with period $T_c$) fetch the percentage of the computational resource pool currently not in use from all the neighboring edge devices (lines 7-10, Algorithm 1). The current periodic fetching can be easily modified to have it event-triggered instead.
Task Placer

The task placer’s role in ORCH is to decide on which edge device the incoming task should be placed for execution. In order to do this, it takes as input the tagged tasks coming from the load characterizer, as well as information on current capacity coming from the capacity estimator. A deadline-aware closest-first task placement strategy presented in Function 2 is used.

**Function 2: Deadline-aware closest-first task placement**

```
parameter: τ_a // Available capacity threshold
input : r, tag, e // Task, tag, receiving edge
output : device // Placement for r

Function PlaceTask(r, tag, e)

    // Try to place r on receiving edge device e
    if GetAvailCapacity(e) ≥ τ_a then
        if Not(tag = DT and e is mobile) then
            Place r on e;
        else
            // Try to place r on one neighboring edge device
            N ← GetNeighboringEdgeDevices(e);
            foreach n ∈ N do
                if GetAvailCapacity(n) ≥ τ_a then
                    if Not(tag = DT and n is mobile) then
                        Place r on n;
                else if No placement found at the edge and tag = DT then
                    Place r on the cloud;
            else
                Failure to place r;
```

The general idea of the function is to place the tasks spatially as close as possible to the requesting end device in order to avoid unnecessary communication overhead. Hence, the function will try placing the task:

1) on the receiving edge device, which is the closest edge device and the entry point for any end device request,

2) on a neighboring edge device if the closest device (in space) does not have enough resources according to the capacity estimator,

3) on the cloud otherwise.

In this study, the considered neighboring devices are an edge device able to serve the whole area if the receiving device is an edge device serving only one
4. DISTRIBUTED ORCHESTRATION USING MOBILE EDGE DEVICES

segment, but this could be extended, for example to allow for more placement attempts.

The only exception to the above is when the task is of category DT and the receiving/neighboring edge device is a mobile edge device (line 3/line 9). As mobile edge resources are scarce and are the only ones capable of handling DS tasks, they should be preferably used for this task type. DT tasks are therefore only placed on stationary edge devices or cloud devices.

Once a placement decision has been taken, the task is transmitted to the executing device, which may or may not be the same as the receiving device. When getting an execution request, the edge device puts it directly on its execution list.

4.4.4 Area Orchestration

The area orchestration proposal is provided in Algorithm 3. It shows the high-level logic of the area orchestration running on the edge devices selected to be area orchestrators, one per edge orchestration area.

The load change detector runs periodically (with detection period $T_d$) but this could be modified to have event-triggered change detection. Each area orchestrator also keeps track of the current high load segments as a set $\mathcal{H}$ of ordered segments and of the current placement of the mobile edge devices as a list $P$ of $<e,s>$ for each $e \in \mathcal{E}_{mob}$.

When a load change is detected, the mobile edge placer is called to place the available mobile edge devices. After the mobile edge placement is performed, the edge devices move to their new locations if relevant. Details of this mechanism are outside the scope of this work.

The area orchestration proposal presented in this section is what will be used in ORCH as a strategy for the edge mobility described in Section 4.3.6.

Algorithm 3: Area orchestration

```plaintext
Algorithm 3: Area orchestration

parameter: $T_d$ // Detection period
state : $\mathcal{H}$ // High load segments
        $P$ // Mobile edge placement
// Runs at each area orchestrator $o$
1 every $T_d$ do
   // Load change detector
2    <isChanging,$\mathcal{H}$> ← LoadChangeDetected($o$, $\mathcal{H}$);
3    if isChanging then
   // Mobile edge placer
4        newPlace ← PlaceMobileEdge($o$, $\mathcal{H}$, $P$);
5        Instruct devices to move when needed;
6        Update $P$ with newPlace;
```

50
4.4. ORCH Framework

Load Change Detector

The load change detector is described in Procedure 4 and has four parts: a load aggregation part, a prediction part, a high load identification part, and a change detection part.

The load aggregation part (lines 4-6) fetches the load information stored as part of the edge orchestration at every edge device during the previous detection period to determine the load per segment in the whole area. In order to avoid unnecessary use of memory, this information is deleted once fetch by the load change detector.

Procedure 4: Load change detection including prediction

```
parameter: \( \tau_h \) // High load threshold
input : \( o \) // Area orchestrator
        \( curH \) // Current high load segments
output : \( isChanging \) // Detection result
        \( hlSegs \) // New high load segments
state : \( areaL = <<< s, \#DS >> \) // \# DS tasks per segment \( s \) in the area

Procedure LoadChangeDetected(\( o, curH \))
1 \( S \leftarrow \text{GetSegmentsInArea}(o) \);
2 \( E \leftarrow \text{GetEdgeDevicesInArea}(o) \);
// Aggregate the DS load per segment
3 foreach \( e \) in \( E \) do
4     foreach \( s \) in \( S \) do
5         Increment \( \#DS \) for \( s \) in \( areaL \) by GetLoadInfo(\( e, s \));
6 foreach \( s \) in \( S \) do
7     // Predict DS load
8     \( pred \leftarrow \text{FUSD_Prediction}(s, \#DS \text{ for } s \text{ from } areaL) \);
9     // Identify high load segments
10    if \( pred \geq \tau_h \) then
11        Insert \( s \) to \( hlSegs \) so that segments are ordered with highest load first;
12 // Detect load change
13 if \( hlSegs \neq curH \) then
14    \( isChanging \leftarrow \text{true}; \)
```

For the prediction part (line 8), an adaptation of the Fast Up and Slow Down (FUSD) algorithm proposed by Xiao et al. [130] is used. Instead of predicting the percentage of CPU utilization, the FUSD algorithm is used to predict the estimated number of DS tasks in each segment in the next detection period. The predictor keeps load and prediction information for a certain number of previous periods, called the window \( W \). In order for the
prediction to capture the increase and decrease of the resource need in a satisfactory way, one parameter is used when the load is increasing (\( \alpha \)) and another when the load is decreasing (\( -\alpha \)).

For identification (lines 9-10), a similar approach to the hot spot classification by Xiao et al. is used. Segments with high predicted load are identified using a threshold \( \tau \), and sorted with highest load first. Finally, a load change is detected (lines 11-12) if the set of ordered high load segments has changed since the previous detection period.

It is part of future work to use adaptive ways of performing the load change detection, e.g., with machine learning.

Mobile Edge Placer

The second area orchestration component focuses on serving as many DS tasks as possible. It considers the ordered set of segments with high DS load that was computed in the load change detector as the segments most in need of a mobile edge. The general idea is then to associate the segment with highest load with the mobile edge device currently located in the segment with the lowest DS load, until no more mobile edge devices are available. This is performed in three steps presented in Function 5.

**Function 5: Deadline-aware mobile edge placement**

```plaintext
input : o // Area orchestrator
       curH // Current high load segments
       curP // Current mobile edge placement
output: newP // New mobile edge placement

Function PlaceMobileEdge(o, curH, curP)

1. availME ← GetSortedAvailableMobileEdge(o);
   // Keep existing placements as much as possible
2. foreach s in the first \(|\text{availME}|\) high load segments do
3. if s is already associated to e in curP then
4.   Add <e, s> to newP;
5.   Remove e from availME and s from curH;
6.   // Associate remaining edge devices with remaining high load segments
7. foreach s ∈ curH do
8.   if there is a remaining mobile edge device then
9.     Add <Head availME), s> to newP;
10.    Remove Head availME) from availME;
```

First (line 2), the available mobile edge devices, i.e., those not in the moving state, ordered by lowest DS load at the current segment are fetched. The
number of currently available mobile edge devices (denoted as \(|\text{availME}|\)) will constrain how many high load segments the ORCH framework can serve.

Then (lines 3-6), the mobile edge placer determines if some of the first \(|\text{availME}|\) high load segments (those with highest load) are already associated with an edge device in the current placement. Those associations are kept to avoid unnecessary unavailable periods due to mobile edge movements, since a mobile edge device is unavailable when moving.

Finally (lines 7-10), the remaining mobile edge devices, if any, are associated with the remaining high load segments.

4.5 Evaluation

In this section, we present the simulation environment used to evaluate the ORCH framework. The evaluation consists in comparing the performance of the ORCH framework with the proposed algorithms and functions to three baseline strategies. The comparison is made in terms of QoS and edge utilization in presence of different loads.

4.5.1 ORCH Implementation

The ORCH framework is evaluated by extending EdgeCloudSim [131], an edge environment simulation tool built upon CloudSim [132]. The tool was adapted to the models used in this work by implementing the mobility of edge devices, the area model with segments, and the ORCH framework structure with corresponding algorithms and functions, among other minor adaptations such as the possibility of end devices leaving the experiment area and having the base unit of simulation as the millisecond instead of the second.

The Java code for ORCH is available online\(^1\). The extension of EdgeCloudSim includes 6 new abstract classes (one for the ORCH orchestrator and five for the ORCH framework components) that allow for modularity, and the corresponding 6 classes corresponding to the implementation of the proposal. Other variants of the ORCH components can thus easily be implemented by implementing new concrete classes for the components. Moreover, 3 additional classes were implemented corresponding to the models for the load generation, the end device mobility, and the edge device mobility. Two additional setup files were created (for the area model and the application model) as new variations of the original EdgeCloudSim simulator. The different parameters used in the experiments are added to the parameter file for easy modifications when conducting experiments.

\(^1\)https://gitlab.liu.se/ida-rtslab/public-code/2019_orch
Building a distributed mobile edge extension to EdgeCloudSim was facilitated by its modular design. However, implementing and integrating the new models and the new scenarios was a challenge that required a deep understanding of both EdgeCloudSim and CloudSim, due to complex interactions. Moreover, implementing the ORCH framework in a modular way to ease testing new variants of the components or experimenting with alternative models added to the implementation complexity.

According to our experiments, scaling up and down the size of the experiment area and of the edge infrastructure is relatively straightforward and does not have a strong impact on the simulator running time, but the total number of tasks generated by the end devices does. The maximum number of tasks submitted in the experiments described later in this chapter was 155458 and the running time for this run was 1 min 50 sec. On average, the simulator running time is less than 1 min 30 sec when simulating 20 minutes of experiment.

4.5.2 Performance Metrics

The task and edge placement outcome is evaluated using a metric that reflects success or failure of a task completing in time denoted as completed tasks below. More specifically:

- success ratio per task category
- number of completed tasks per device type

Looking at the outcome per task category is important given that a high QoS requires that the success ratio for DS tasks should be as high as possible. Indeed, the deadline-aware approach to the D2TEP problem presented in this study focuses on serving the DS tasks with mobile edge resources.

The outcome per device type shows on which device type the completed tasks were executed. The number of tasks completed on edge devices should be high (especially on mobile ones) and the number of tasks completed in the cloud should be low. This is natural as it was the very motivation for using edge devices and mobile ones in particular, to create added QoS compared to a classic cloud.

4.5.3 Baseline Strategies

The strategies for task placement and edge mobility presented in Section 4.4 and implemented in the ORCH framework are compared to four baselines strategies: one for task placement and three for edge mobility.
4.5. Evaluation

Task Placement

The ORCH task placement strategy is compared to a First Fit strategy. In this strategy, the task is placed on the receiving edge device if resources suffice, otherwise on the neighbouring edge device if enough resources and otherwise on the cloud. First Fit strategies are commonly used in recent research as a baseline for task placement strategies [95, 99, 94].

Edge Mobility

The three baselines used for edge mobility are the following: 1) a strategy where all edge devices are stationary, called Stationary, 2) a perfect edge mobility strategy, called Perfect, and 3) a simpler edge mobility strategy, called Scheduled.

For all four strategies the total amount of resources (i.e. CPU available) is the same. The resources are equally distributed among stationary edge devices serving the whole area (called area-wide) and edge devices serving one segment each (called segment-wide). For the Stationary strategy, the segment-wide edge devices are stationary, whereas they are mobile for the Scheduled, Perfect and ORCH strategies.

The Stationary strategy corresponds to the current research in edge/fog resource management, where the mobility of edge devices is not considered. In the Perfect strategy, it is assumed that some mobile edge device exactly follows the movements of an end device running urgent applications, as long as there are mobile edge devices available. The Perfect strategy is therefore an unrealistic one, as it would imply a perfect knowledge of end user mobility at all time. However, it gives a measure of the maximal performance that the system could theoretically achieve.

In the Scheduled strategy, the mobile edge devices move from one segment to another in a pre-defined pattern (e.g. as an edge device in a scheduled bus might do, with bus stops acting as serving positions). In the following experiments, they move clockwise around the orchestration area, starting from a random segment. Büchel and Corman [133] showed that the best fitting distribution for stop duration at bus stops and traveling time between bus stops is a lognormal distribution, based on empirical data. Therefore, in the experiments, the stop duration (in seconds) at a serving position is randomly selected from a lognormal distribution, with parameters $\mu = 2.5$ and $\sigma^2 = 0.49$. Similarly, the time in between two neighboring serving positions (in seconds) is randomly selected from a different lognormal distribution, with parameters $\mu = 3.3$ and $\sigma^2 = 0.04$.

4.5.4 Experimental Setup

A neighborhood composed of one outside segment and one edge orchestration area with eighteen segments is considered for the evaluation in this
study. The edge resources consist of one area-wide device and two segment-wide devices. On the resource demand side, we consider 25 end devices (that could also be thought of as group of end devices moving in the same way). The experimental setup can be changed to increase the scale of the edge infrastructure (at the cost of increased simulation time) but the chosen scale is similar to state-of-the-art service placement research [94, 95, 98].

The neighborhood is illustrated in Figure 4.4. The edge orchestration area is composed of segments being represented as blue shapes. Half of the segments are residential segments (light dashed shapes) and the rest of the segments are commercial segments (dark colored shapes). The serving positions are denoted with dark blue circles, and the outside segment is represented in light yellow.

For each run of all four strategies, the movement of each end device is randomly generated and some end devices are randomly selected for running urgent applications. All end devices create tasks following the load model of Section 4.3.4 if they are located in a residential or commercial segment. This creates different surge locations and duration for every run.

Table 4.1 summarizes all the parameters and corresponding values considered in the simulation. The simulation uses milliseconds as a unit of time, contrary to previous work using seconds [95, 131]. This reflects the edge context better. In particular, communication delays in 5G are in the order of a few ms [134], which is reflected in $d_{short}$. Changing the values for communication delays between different device types may impact the results. However, the aim of this study was not to optimize the round trip time in a specific scenario, but rather to show that a higher success ratio is achieved when using the ORCH mobile edge concept.
### 4.5. Evaluation

Table 4.1: Simulation parameters.

| Area                  | Parameter                                                                 | Value                        |
|-----------------------|---------------------------------------------------------------------------|------------------------------|
| Generic               | Simulated time # end devices \((M)\)                                       | 1200000 ms (20 min)          |
|                       |                                                                           | 25                           |
| Computation           | Edge device                                                              | 500 MIPS                     |
|                       | Cloud device                                                             | 12000 MIPS                   |
|                       | CPU share per task                                                       | 20%                          |
| Communication         | \(d_{\text{short}}\)                                                     | 1 ms                         |
|                       | \(d_{\text{medium}}\)                                                   | 25 ms                        |
|                       | \(d_{\text{long}}\)                                                     | 100 ms                       |
|                       | Computation req. \((R_r)\)                                               | 10000 MI                     |
|                       | DS task deadline \((D_r)\)                                               | 30 ms                        |
|                       | DC task deadline \((D_r)\)                                               | 150 ms                       |
|                       | DT task deadline \((D_r)\)                                               | 300 ms                       |
|                       | Urgent applications Task mix \((DS/DC/DT)\)                              | 40%/50%/10%                  |
|                       | Poisson mean                                                             | 50 ms                        |
|                       | Regular applications Task mix \((DS/DC/DT)\)                            | 0%/70%/30%                   |
|                       | Poisson mean                                                             | 500 ms                       |
| Segments              | Rate \(\lambda^5\)                                                      | 36000 ms                     |
|                       | Rate \(\lambda^1\)                                                      | 300000 ms                    |
| Edge mobility         | Scheduled: stable state duration distribution                           | Lognormal(\(\mu = 2.5,\)
|                       | Scheduled: moving state duration distr.                                  | \(\sigma^2 = 0.49\))        |
|                       | ORCH: moving state duration distribution                                  | Lognormal(\(\mu = 3.3,\)
|                       |                                                                         | \(\sigma^2 = 0.04\))        |
|                       |                                                                         | Gamma(\(k = 7.5, \theta = 4\))|
| Capacity estimator    | Period \(T_c\)                                                           | 1000 ms                      |
| Task placer           | Threshold \(\tau_a\)                                                    | 5                            |
| Load change detector  | Period \(T_d\)                                                           | 1000 ms                      |
|                       | Window size \(W\)                                                       | 8                            |
|                       | FUSD percentile                                                          | 90th                         |
|                       | \(\uparrow \alpha\)                                                     | -0.2                         |
|                       | \(\downarrow \alpha\)                                                   | 0.7                          |
|                       | Threshold \(\tau_h\)                                                    | 5                            |
4. DISTRIBUTED ORCHESTRATION USING MOBILE EDGE DEVICES

4.5.5 Load Scenarios

When evaluating the system, we consider two load scenarios. In the first one, the number of end devices running urgent applications is the same as the number of available segment-wide edge devices (i.e. 2). Hence the load is balanced with regards to resources but the load surge varies in space over time. In the second scenario, the number of end devices running urgent applications (set to 6) exceeds the number of available segment-wide edge devices, hence creating a system level overload.

4.5.6 Impact of the Task Placement Strategy

To isolate the impact of the task placement strategy, we fix the edge placement strategy to Perfect for the following experiment. We perform 50 runs per task placement strategy.

Balanced Load Scenario

Figure 4.5a shows the average success ratios for DS tasks, in the balanced load scenario. It is similar for both task placement strategies (ORCH and First Fit) and close to 100% as the Perfect edge mobility strategy ensures that the mobile edge devices are following the devices running urgent applications. Therefore, it appears that the ORCH task placement does not add to QoS for DS tasks. However, when looking at the average percentage of failed tasks that are due to a deadline miss (shown on Figure 4.5b), it is 57% for the First Fit and 0% for ORCH. This means that the ORCH strategy is able to avoid wasting resources by executing tasks on devices when those tasks are deemed to miss their deadline. This is possible because the task placer in ORCH knows the type of the tasks thanks to the task characterizer and can avoid placements that will for sure lead to a deadline miss.

Figure 4.5: Impact of task placement strategy (balanced load).
4.5. Evaluation

(a) on success ratios for DS tasks

(b) on failures due to deadline miss

Figure 4.6: Impact of task placement strategy (overload).

Overload Scenario

Figure 4.6 shows the same metrics as Figure 4.5 but for the overload scenario. Regarding the average success ratios for DS tasks, the results in the overload scenario (shown on Figure 4.6a) are very similar to the ones for the balanced scenario, i.e. a similar percentage for both the First Fit strategy and the ORCH one, but now the actual ratio is a lot lower, due to the overload that makes that not all DS tasks can be handled, even with a Perfect edge mobility. Therefore the conclusion that the ORCH task placement does not add to QoS for DS tasks is also valid in the overload scenario. If looking at the average percentage of failed tasks (shown on Figure 4.6b), we can see that now almost all failures of the First Fit scenario (99.8%) are due to a deadline miss, but still none for the ORCH task placement. Thus, the fact that the ORCH strategy successfully avoid wasting resources is even more visible in the overload scenario.

4.5.7 Impact of the Edge Placement Strategy

Then, we study the impact of the edge placement strategy by performing 50 runs per edge placement strategy. For this experiment, we use the ORCH task placement strategy in all edge devices.

Balanced Load Scenario

Figure 4.7 shows the success ratios of DS tasks for the balanced load study with all the baselines. Key indicators for this study are shown in Table 4.2. Regarding the DS tasks, ORCH outperforms Scheduled and Stationary with a 13 times, respectively 4 times higher average success ratio for DS tasks (83.3% vs 6.3%, respectively 17.8%). ORCH also comes as close as 83% to the Perfect performance which succeeds for 99.9% of the DS tasks.
Table 4.2: Average, standard deviation, minimum, and maximum number of successful DS tasks for the balanced load (B) and overload (O) scenarios.

| Strategy | Average | σ  | Min | Max |
|----------|---------|----|-----|-----|
|          | B       | O  | B   | O   |
| Stationary | 2097    | 4681 | 2418 | 4189 | 0   | 0   | 8831 | 15366 |
| Scheduled  | 751     | 2362 | 381  | 600  | 99  | 1345 | 1747 | 3851  |
| ORCH      | 10103   | 19854 | 3356 | 4639 | 2934 | 9416 | 16195 | 28305 |
| Perfect   | 11881   | 21816 | 3336 | 5967 | 4097 | 12580 | 17099 | 31660 |

Figure 4.7: Success ratios for DS tasks (balanced load).

Regarding the DC and DT tasks, the results (shown in Figure 4.8) are similar for the four strategies, with a success ratio very close or equal to 100%. This is expected as all strategies handle those tasks in a similar way.

Figure 4.9 shows the second metric of evaluation: average number of tasks completed on each edge device type in the balanced scenario. The number of tasks completed in the cloud is negligible for all four strategies (the resources of edge devices are enough to handle the load) and is therefore not shown. The strategies can be divided into two groups. First, Stationary and Scheduled have a minority (15%, respectively 6%) of tasks completed in the segment-wide devices. On the opposite, Perfect and ORCH exhibit a balanced division with 43%, respectively 47% of tasks completed in the segment-wide devices. A balanced division is good because the segment-wide devices have a higher utilization (less waste of resource) and the area-
4.5. Evaluation

(a) DC tasks

(b) DT tasks

Figure 4.8: Success ratios (balanced load).

Figure 4.9: Division of completed tasks among mobile/stationary edge devices (balanced load).

wide ones have more free resources for handling a higher load without having to resort to cloud resources.

Overload Scenario

Figure 4.10 shows the success ratios of DS tasks for the different edge placement strategies with key indicators summarized in Table 4.2.

In this highly overloaded scenario where there are potentially three times more demand than supply, the success ratio for DS tasks is now on average
12.6% for the Stationary strategy, 6.3% for the Scheduled strategy, 54.1% for the ORCH strategy, and 58.4% for the Perfect strategy. The results indicate that while Scheduled and Stationary continue to be incapable to manage the load, ORCH handles 92% of the Perfect service rate in this overly loaded scenario. Moreover, ORCH handles 8 times more DS tasks than Scheduled on average and always at least 144% more tasks than the highest Scheduled performance. The average success ratios for DC and DT tasks are similar to the previous ones, with a slight decrease for the DC tasks (at most 2%, shown on Figure 4.11).
With regard to the division of completed tasks among device types, the results for the overload scenario were comparable to those for the balanced load one, as it can be seen on Figure 4.12.

4.6 Discussion

In this study, we presented the ORCH framework as a mean to solve the D2TEP problem and evaluated one possible implementation of the framework. However, it should be kept in mind that it is a first proposal, with relatively straightforward algorithms for each component. Nevertheless, when those are combined within the ORCH framework, they manage to achieve high QoS and edge utilization. Therefore, the ORCH framework is promising and improvements of its different components should be considered.

Regarding task placement, a next step is to include task migration as part of the task placer role, so that the edge device can modify the current task placement dynamically when receiving a new task, in case this would improve the QoS or the edge utilization.

Regarding edge placement, DS tasks are mostly rejected when there is no available segment-wide edge device. This either happens because the segment-wide edge devices are moving between serving positions (Scheduled/ORCH), or they are not at the surge location, either because of their pre-defined schedule/fixed location (Scheduled/Stationary) or because the
4. DISTRIBUTED ORCHESTRATION USING MOBILE EDGE DEVICES

load change has not yet been detected (ORCH). The reported results where ORCH outperforms Scheduled and Stationary show the importance of having mobile segment-wide devices as well as intelligent placing strategies, so that they are available as much as possible and where they are needed.

Moreover, the results vary (see the standard deviation in Table 4.2) depending on how the load for the actual run looks like, i.e. whether end devices generating DS traffic are in the same segment or are spread all over the orchestration area, and whether those end devices stay in the orchestration area during the whole experiment or are subject to churn. Further improvement of the results is therefore possible through adoption of adaptive algorithms.

In a nutshell, using a mix of mobile and stationary edge devices provides the ability to serve all edge task categories in presence of local sudden surges in load in a flexible manner, i.e. without having to change the infrastructure in place. This is especially interesting when the delay-sensitive traffic is present.
In this chapter, the focus is on the resource demand aspect. We choose as application domain mixed reality and implement our own MR edge application. Then, we study its performance and resource usage for different configurations of the communication link and of hardware used.

5.1 Scenario

In this study, we take a MR application as described in Section 2.2.2 and make it work at the edge by offloading parts of it. We decide to offload to the edge the second and third parts (i.e. point cloud creation and graphic overlay creation) of the MR application pipeline shown in Figure 2.6, as they are candidates for resulting in heavy computation and/or heavy data transmission. The point cloud creation part is always computationally intensive, as it requires complex computer vision algorithms, whereas the graphic overlay creation may not be. However, the graphic overlay creation requires the point cloud knowledge in order to add the graphics correctly, meaning at the thought location in the point cloud. As the point cloud data is quite important and modified all the time, it is not desirable to transmit it frequently. The other parts of the MR pipeline (sensor input and display) are kept in the end device. Hence, the generic offloading scenario presented in Figure 3.2 will look like Figure 5.1 in our MR case study.

We now dive deeper into the different steps that are involved in the above scenario. First, our application allows two types of sensor input. The first one is a video input that will be used for creating the point cloud, and the second one is a user input, as the application allows the user to add or remove virtual elements from the scene. This is done by pressing a button on the screen.
5. UNDERSTANDING MIXED REALITY OFFLOADING WITH EDGE COMPUTING

Therefore, the application will have two workflows, that are however mostly similar.

We start by describing the workflow triggered by a video input. This workflow is illustrated in Figure 5.2. In this case, the scenario consists in capturing a dynamic scene with a smartphone and sending the captured video stream to an edge device that will analyze it to create a point cloud. The scene is dynamic because the camera is moving. Then, the stream is augmented with a visualization of the point cloud sent back to the smartphone. We can see that in addition to the steps present in Figure 2.6, additional steps are required. Those are related to the video transmission between the end and the edge devices and consist into encoding what needs to be transmitted, choosing a protocol for the transmission and decoding it upon arrival at the destination device. This needs to be done for the uplink transmission (from the end device to the edge device) and also for the downlink transmission (from the edge device to the end device).

In the case of a user input (adding or removing a virtual element to or from the scene), only steps a) and c) are modified. It is illustrated with green boxes in Figure 5.3. The main difference is that the uplink transmission now only consists of a message indicating the element to be added, and not of the full video stream.

5.2 MR-Leo: a Modular Prototype

To study of the feasibility and performance of offloading to the edge for an MR application, we developed our own prototype, called MR-Leo1. This is to the best of our knowledge the first available open-source prototype for

\(^{1}\)For Mixed Reality - Linköping edge offloading.
5.2. MR-Leo: a Modular Prototype

Figure 5.2: MR workflow for video input.

Figure 5.3: MR workflow for user input.

MR that allows video streaming back to the end device and in-depth study of the communication link.

In the following sections, we present the architecture of the prototype, some design choices and discuss insights gained during the implementation process. The code for the prototype is made fully available online\(^2\)\(^,\)\(^3\) in the hope to contribute to further development and studies of MR. In addition to the code, readers interested in implementation details are referred to [135].

\(^2\)https://gitlab.liu.se/ida-rtslab/public-code/2019_mrleo_client
\(^3\)https://gitlab.liu.se/ida-rtslab/public-code/2019_mrleo_server
5. UNDERSTANDING MIXED REALITY OFFLOADING WITH EDGE COMPUTING

5.2.1 Architecture

MR-Leo has a client/server architecture. On the client side (the end device), only the input/output tasks of the MR pipeline presented in Figure 2.6 are performed. On the server side (the edge device), the heavy computation part of the MR pipeline is performed. This corresponds to what is shown in Figure 5.1.

To implement the communication link that will transport the required data between the end and the edge devices (yellow arrows in Figures 5.2 and 5.3), two major aspects had to be dealt with. First, the encoding and decoding of the video stream on both the end and the edge device. Second, the choice of a transmission protocol for the video stream. As we want to study different alternatives for those parts, the architecture is made modular to allow easy switching between different variants for those components.

Figure 5.4 presents a high-level view of the MR-Leo architecture. The client part takes the two types of input already mentioned: video input (either coming from the end device’s camera or a recording) and user input (e.g. pressing the screen to add a MR element). The client outputs via the user interface both the original video stream and the resulting MR-enhanced video stream on the end device display. Figure 5.5 shows a screenshot of the user interface. The video streamer and the video receiver are in charge of preparing the video for transmission, respectively reception over the communication link (including encoding, respectively decoding).

On the server part of MR-Leo, there are two kinds of input possible, both coming from the client part: the encoded video stream and messages indicat-
5.2. MR-Leo: a Modular Prototype

Figure 5.5: Screenshot of the MR-Leo client

ing user input. The server outputs a MR-enhanced video stream to the client. Mirrored from the client, the arriving video stream has to be decoded and the MR-enhanced stream is encoded before leaving the server. The central part of the server software is the MR component, where the heavy computation required for the point cloud and the graphic overlay creation is performed.

The MR-Leo architecture is modular, making it easy to test different networking protocols, encoders/decoders or MR frameworks (for example other implementations of SLAM techniques such as CNN-SLAM [136] or LSD-SLAM[137]).

5.2.2 Design Choices

MR-Leo was implemented to work on an Android smartphone acting as the client and a Linux machine acting as the server. To implement the MR-Leo architecture, we use a mix of already existing software frameworks/libraries and custom-implemented components, when off-the-shelf alternatives were not suitable or available. Figure 5.4 also indicates the parts we implemented completely ourselves (in blue), the parts where we reused existing open-source libraries (in red) and the parts for which we modified existing open-source libraries (in purple). Parts that are on one side blue and one the other side red are parts where some configurations reused existing libraries and other configurations were implemented by us.

The open-source frameworks that were used were:

- Gstreamer\(^4\) for implementing the video streaming/receiving parts.
- ORB-SLAM2 [138] for implementing SLAM techniques for MR.
- Pangolin\(^5\) for rendering MR graphics to an image feed.

It was out of the scope of this work to evaluate the performance of the SLAM algorithms used. Therefore, we selected the ORB-SLAM2 framework that obtained good results in the SLAM benchmark SLAMBench2 [139], as a representative for state-of-the-art SLAM framework. For the server video

\(^4\)https://gstreamer.freedesktop.org/  
\(^5\)https://github.com/stevenlovegrove/Pangolin
streamer and the client video receiver, we use Gstreamer or a custom implementation of an MJPEG encoder/decoder as alternatives in our evaluations below.

### 5.2.3 Insights

During the implementation of MR-Leo, we gained several important insights about what to consider when developing a MR application. We describe them here as they might be guidelines for further work in this area.

Currently, the SLAM algorithms used at the edge for creating the point cloud do not have a bounded execution time. This means that it can take variable time analyzing a specific frame. During that time, the next frames cannot be analyzed and are queued in the server. In order to avoid latency accumulation, a resilient mechanism has been implemented to ensure that outdated frames will not be handled by the application and are dropped. However, this should not be happening too often in order to keep the QoS. Therefore, one guideline for developing a responsive MR application is to strive towards handling a frame before the next one comes.

Although the communication between the end and the edge devices and the MR calculations may appear as being two separate parts of offloading to the edge, they are in reality intertwined. Indeed, what is chosen as communication link will have an impact on the quality of the video stream to be used as input for the MR algorithm. If this quality is too low for the chosen MR algorithms, then the QoS of the application will degrade, not because of the choices made for MR but due to the communication link.

Some important benefits of taking off-the-shelf components for implementing MR-Leo are: reduced developing time, access to a community and less frequent (or patchable) bugs. However, this choice also comes with important drawbacks in the context of MR. We noticed that in order to achieve the low latency which was our target for MR at the edge, one has to optimize both the MR framework and create tailored streaming components that are more specialized than the current ones.

Finally, encoders/decoders used are sensitive to the frame rate of the video stream and irregular frame arrivals will have a negative effect on performance. Therefore, another guideline for developing a MR application is to reduce the difference in time spent for analysis of each frame, so that the MR output is as regular as possible.

### 5.3 Understanding Latency and Throughput

We present first the experiments performed and the test environment in which they were conducted. Then, we discuss the used metrics and in particular the concept of end-to-end latency for mixed reality.
5.3. Understanding Latency and Throughput

5.3.1 Experiments Conducted

The evaluations consist in comparing different configurations that can be used for the scenario presented in Section 5.1. The configurable parts compared were:

i) the video compression format used for encoding at the end device (and necessarily, for decoding at the edge device), i.e. (b) and (d) in Figures 5.2 and 5.3.

ii) the transport protocol used for the uplink transmission, i.e. (c).

iii) the video compression format for encoding at the edge device (and necessarily, for decoding at the end device), i.e. (g) and (i).

iv) the transport protocol used for the downlink transmission, i.e. (h).

The other parts of the prototype can be modified but this was out of the scope.

Each experiment was conducted 30 times for the same configuration in order to mask any network interference or computing hardware performance fluctuations. In order to ensure reproducibility between the runs, a video play-back is used instead of the actual camera feed and the addition of a virtual element was automatically triggered five times during the video, every 10 seconds. The test video used is 60 seconds long and is set up in an indoor environment. Figure 5.6 shows an extract of it. The full video is available online\(^6\), has a resolution of 640x480 pixels and a frame rate of 30 fps.

![Screenshot of the test video.](https://gitlab.liu.se/ida-rtslab/public-code/2019_mrleo_video)

---

\(^6\)https://gitlab.liu.se/ida-rtslab/public-code/2019_mrleo_video
5. UNDERSTANDING MIXED REALITY OFFLOADING WITH EDGE COMPUTING

5.3.2 Test Environment

We first ran the experiments with a first edge/end device pair, hereafter called the baseline devices. Then, we conducted a second set of experiences with another edge device or end device that exhibit different characteristics of interest. Those are denoted as the extra devices.

Regarding edge devices, the baseline device is a Lenovo Thinkpad T450s laptop. The laptop runs Ubuntu 18.04 and has 12 GB RAM and an Intel Core i5-5200U CPU (2.2 GHz, 2 cores, 4 threads), which is classified as a high mid-range CPU as of July 2019 [140]. The extra edge device is an HP Elitebook 840 G5. It runs Ubuntu 18.10 and has 16 GB RAM and an Intel Core i7-8550U CPU (1.8 GHz, 4 cores, 8 threads), which is classified as a high end CPU as of July 2019 [141]. Therefore, the extra edge device is expected to be somewhat more powerful than the baseline one.

The baseline end device is a LG G6 smartphone running Android 8.0 and equipped with the Qualcomm Snapdragon 821 mobile platform. It contains a Qualcomm Kryo CPU (2.4 GHz, 4 cores) and 4 GB RAM. The extra end device is a Samsung A5 (2017 model) also running Android 8.0 and equipped with the Samsung Exynos 7880 mobile processor. It contains a Cortex A53 CPU (1.9 GHz, 8 cores) and 3 GB RAM. Compared to the baseline device it is a bit less powerful but provided hardware from a different vendor.

During the experiments, all third-party applications on the end devices were uninstalled, and the internal applications that could be disabled were disabled. On the edge devices, no other application, scheduled tasks or other services were running in the background.

The experiments were performed over a local network set up using an Asus RT-N12 router disconnected from the Internet. The edge device was connected to the network using an Ethernet cable and Gigabit Ethernet (1000 Mbit/sec), and the end device was connected using an 802.11n wireless network. The end and edge devices were placed within one meter from the network gateway, and the same positions were used for all tests using the same devices. Figure 5.7 shows a picture of the test environment.

5.3.3 Performance Metrics

In order to measure the performance of the MR-Leo implementation in the different configurations, we study two different aspects. Both are of importance so that the virtual elements integrate seamlessly with the reality, i.e. for QoS. The first aspect is the latency of the application. This one should be low in order for the application to be reactive enough to the user’s movement and interactions. The second aspect is the throughput of the application, i.e. how much of the incoming stream is displayed with the added virtual elements to the user. This contributes to QoS by being related to how well the virtual
5.3. Understanding Latency and Throughput

![Test environment](image)

Figure 5.7: Test environment.

elements integrate with the reality in a seamless manner and should be as high as possible.

For the latency aspect, the metric commonly used [27] is end-to-end (E2E) latency. However, in the context of this study, this could mean two different things. The first way to consider E2E latency is to measure at the end device the time elapsed between the moment the end user presses the "Add virtual element" button and the moment when the virtual element appears on the display (Steps (a-bis) to (j) on Figure 5.3). We denote this first latency metric as the **time to virtual element** (T2VE). This is the metric considered for E2E latency by Chen et al. [27].

However, we argue that in our MR scenario, E2E latency could also be understood and measured as the time it takes for a captured video frame to be displayed with the MR enhancement on the display (Steps (a) to (j) on Figure 5.2). In our scenario, this is expected to be different from the T2VE as the video has to be transmitted to the edge device and back, and not only one way. We denote this second latency metric as the **frame round trip time** (FRTT).

The T2VE is automatically measured by adding in the mixed reality part, in addition to the virtual element, a row of pixels with a uniform color not present in the video and detecting in the end device the appearance of this row. Our tests showed that this detection is lightweight and do not impact the results. Similarly, we modify some frames of the original video to contain a row of pixel with a second color for the FRTT measurements.

The throughput aspect corresponds to how many of the incoming frames are processed by the edge device and how many are discarded due to the resilient mechanism described in Section 5.2.2. Indeed, in order for the SLAM algorithms to perform well, the number of frames dropped should be as low as possible. We measure this second aspect as the number of frames received...
5. UNDERSTANDING MIXED REALITY OFFLOADING WITH EDGE COMPUTING

at the end device. This way, we account for frames loss in the MR algorithms, but also during transmission, which is part of the prototype under evaluation.

5.4 First Attempt at Edge Offloading

The baseline configuration consists of the baseline end device and the baseline edge device using a H.264 video stream transmitted over TCP. Those two were chosen because H.264 is widely used for video streaming, and TCP has been chosen for other MR prototypes [27]. We evaluate this baseline configuration using the performance metrics presented in Section 5.3.3.

5.4.1 Results

![Latency CDFs for the baseline configuration.](image)

Figure 5.8: Latency CDFs for the baseline configuration.

Figure 5.8a shows the cumulative distribution function (CDF) of the T2VE measurements for the baseline configuration. It can be seen that the value at the 90th percentile is 1122 ms, but also that the it can be as low as 208 ms or as high as 4363 ms. Since this data includes the measurements made during the 30 runs for the five times a virtual element is triggered, we performed further analysis to identify what is causing those extreme values.
We find out that in the test video, there is in particular one moment where the SLAM framework has difficulties in constructing the point cloud. This moment happens around 30 seconds, when the camera moves out of the previous scene to film the surroundings. If we isolate the measurements for the virtual element triggered at 30 seconds (presented in Figure 5.8b), we can see that they show a lot higher latency with the 90th percentile at 2313 ms, more than double compared to the whole dataset. This is a direct consequence of the point cloud being lost in the MR system, so the virtual element cannot be placed and rendered until the point cloud is recovered.

On the contrary, Figure 5.8c shows the CDF for the T2VE the next time a virtual element is triggered. At this moment in the video, the point cloud creation is stable, and then the T2VE is 6.8 times lower than for the previous one, with a 90th percentile at 342 ms.

During each experiment run, we measure the FRTT for six different frames, with ten seconds within them. Figure 5.8d shows the CDF for the FRTT measurements. We note two interesting aspects. The first one is that contrary to the T2VE measurements, the FRTT measurements are less spread. This is due to the FRTT measurements occurring when the point cloud is stable. The second aspect is that the FRTT measurements show 1.6 times higher latency at the 90th percentile than the T2VE ones (538 vs 342 ms). This is expected as the "add a virtual element" message is transmitted faster than the video stream itself, in its own data link.

![CDF of the throughput for the baseline configuration.](image)

Finally, Figure 5.9 shows the CDF of the throughput measured as the number of frames per second received at the end device. We can see that the baseline throughput varies but in 95% of the cases is superior or equal to 24 fps.
5. UNDERSTANDING MIXED REALITY OFFLOADING WITH EDGE COMPUTING

|                      | Edge system | Point cloud | Graphics | Transmission |
|----------------------|-------------|-------------|----------|--------------|
| Average time spent   | 5           | 31          | 5        | 448          |
| per frame (ms)       |             |             |          |              |
| Standard deviation   | 7.8         | 6.5         | 1.7      | 45.6         |
| (ms)                 |             |             |          |              |
| % of total FRTT      | 1%          | 6%          | 1%       | 92%          |

Table 5.1: Breakdown of the average FRTT.

5.4.2 Acceptable Performance

In order to evaluate the intrinsic performance of the prototype, we look up for performance guidelines for MR applications in the literature, related to the two aspects of performance investigated.

With regards to the latency aspect, the idea is that the MR enhancement should appear as immediate to the user. In his study of acceptable response time for human-computer interactions, Miller [142] present the limit of 100 ms for the system’s answer to be perceived as immediate. 100 ms is also the higher bound that is considered as the limit after which an online action-based game will be perceived as unplayable [143]. However, the latency is dependent on the end device used. For example, mixed reality projected inside VR glasses requires latency to be under 20 ms in order to prevent motion sickness [24]. In our phone-based scenario, we adopt the threshold for acceptable performance as 100 ms.

With regards to the throughput, the displayed video stream should display a high-enough frame rate so that the user cannot identify individual frames. We consider that an acceptable performance on the end device is a video stream returned from the edge device at 24 fps, which is the standard used in the movie industry.

5.4.3 Discussion

With regards to the thresholds defined above, the baseline configuration achieves acceptable performance for the 90th percentile with regards to throughput but not with regards to latency, whether it is T2VE or FRTT. In order to identify which part of the system is the bottleneck, we perform a breakdown of the average FRTT, shown in Table 5.1.

We can see that the vast majority (92%) of the latency is spent in the transmission. Therefore, we investigate alternatives to the baseline configuration for the video transmission in Section 5.5. Moreover, the latency breakdown indicates that time is spent in the edge system. In particular, frames are queued before being processed by the edge device due to the MR framework being occupied at processing one of the previous frames. Therefore, it is also
5.4. First Attempt at Edge Offloading

Another interesting thing that came out of the baseline configuration evaluation is the fact that the video stream used will have an important impact on the performance of the application. When the point cloud is stable (i.e., measurements at the 20th, 40th and 50th seconds), the T2VE is lower than when the SLAM framework used is having issues.

Finally, the difference in measured latency for T2VE and FRTT shows that when studying the performance of a MR application that offloads both the computing and the rendering it is important to consider both in order to get a correct idea of how the application is performing.

|                | Avg | σ  | 90th | Min | Max |
|----------------|-----|----|------|-----|-----|
| **Graphics**   | 5   | 2  | 6    | 3   | 50  |
| **MR**         | 36  | 7  | 44   | 19  | 131 |
| **Edge**       | 41  | 12 | 58   | 20  | 132 |
| **FRTT**       | 489 | 50 | 538  | 380 | 759 |
| **Graphics**   | 4   | 1  | 4    | 2   | 18  |
| **MR**         | 24  | 4  | 27   | 15  | 93  |
| **Edge**       | 24  | 5  | 27   | 16  | 93  |
| **FRTT**       | 552 | 50 | 635  | 472 | 676 |
| **Graphics**   | 5   | 2  | 6    | 3   | 49  |
| **MR**         | 36  | 7  | 45   | 20  | 133 |
| **Edge**       | 42  | 13 | 61   | 20  | 138 |
| **FRTT**       | 666 | 37 | 705  | 591 | 827 |

Table 5.2: Performance measurements statistics (ms).
5. **Understanding Mixed Reality Offloading with Edge Computing**

5.5 **Focus on the Communication Link**

The prototype was designed so that it is easy to study the impact of changed modules, e.g. the protocol or the video compression format. Previous work focused on the computational part of MR applications. Here, we investigate what could be done in order to decrease the time spent in the communication link, compared to the baseline solution. We study two aspects of the communication link between the end and the edge device: first the transport protocol and then the video compression format.

5.5.1 **Impact of the Transport Protocol**

The transport protocol typically used in related works for transmitting the video frame data is TCP. As described in Section 2.3.2, TCP offers a frame ordering guarantee at the cost of the overhead for the acknowledgment packets. UDP comes with no guarantee but is more lightweight. In this section, we study the impact of replacing TCP with UDP in the baseline configuration used in steps (c) and (h) depicted on Figure 5.2. The rest of the steps are not modified.

We find that changing the transmission protocol has an impact on three aspects of the communication link: the minimum bandwidth required, the latency of the MR application (both T2VE and FRTT), and the application throughput.

As described before, the MR framework requires images of a high enough quality in order to create a point cloud. If the video frames have been compressed so much that interesting features are not distinguishable anymore, then the MR application will not be able to deliver its service. In the prototype, how much the frames from the video stream are compressed is directly related to how much bandwidth is available on the link. Therefore, we performed a separate experiment that step-wise increased the bandwidth available on the communication link and measured the number of feature points tracked by the MR framework. The results are shown in Figure 5.11 and we found that a suitable and stable amount of feature points (around 250) can be achieved for a bandwidth of 2000 kbit/sec for TCP and 4000 kbit/sec for UDP. The difference is due to the fact that UDP has to transmit complete frames all the time because some could be lost whereas TCP only needs to send the difference between frames most of the time. We use those bandwidth values for the latency measurement in order to compare latency and throughput for a similar frame quality used as input to the edge system.

With regards to latency, Figure 5.12a shows the CDF of T2VE measurement for the UDP configuration, with data when the point cloud is stable. At the 90th percentile, the T2VE with UDP is 3.7 times lower than for the baseline using TCP (177 ms vs 654 ms).
5.5. Focus on the Communication Link

Figure 5.11: Number of tracked points depending on transport protocol and bandwidth.

![Graph showing the number of tracked points for TCP and UDP](image)

(a) T2VE, virtual elements added with a stable point cloud.

(b) FRTT

Figure 5.12: Latency CDFs for the UDP configuration.

Similarly, Figure 5.12b shows the CDF for FRTT with UDP. We observe that the behaviour is similar to the one for TCP but the latency is also lower for this metric, with a 90th percentile at 392 ms. However, the gap between TCP and UDP is lower for FRTT: UDP is only 1.4 times faster than TCP for this metric.

With regards to throughput, Figure 5.13a shows that the acceptable frame rate of 24 fps is achieved 79% of the time with UDP, which is lower than for TCP. This is explained by the fact that frame losses are going to be higher with UDP since the protocol has no guarantee for receiving them and this can impact not only the frames loss during the communication but also the performance of the SLAM framework as it may be harder to maintain the point cloud...
5. UNDERSTANDING MIXED REALITY OFFLOADING WITH EDGE COMPUTING

Figure 5.13: Throughput CDFs for different configurations.

Figure 5.14: Latency CDFs for the MJPEG configuration.

cloud if frames are missing (meaning that the computing time per frame at the edge may increase).

5.5.2 Impact of the Video Compression Format

Another interesting part of the communication link is the video compression format, because it will have an effect on how long the encoding/decoding phases will be. As an alternative to the common H.264 format used in the baseline, we use in this section another common format: the MJPEG one, for the steps (g) and (i) of Figure 5.2. No other step is modified and the bandwidth used is the same.

We find that using the MJPEG format improves the latency (both T2VE and FRTT), as shown in Figures 5.14a and 5.14b. The latency at the 90th percentile is indeed 6.7 times lower (98 ms) for T2VE and 2.2 times lower (245 ms) for FRTT.

With regards to the throughput, the MJPEG configuration performs similarly to the H.264 one, with a frame rate superior or equal to 24 fps 96% of
5.6. Impact of Hardware Choices

the time (shown in Figure 5.13b). Since the MR edge part gets exactly the same input as the baseline, this shows that the MJPEG encoding/decoding does not impact the frame rate negatively at the same time as it improves the latency.

5.5.3 Summary of the Communication Link Study

We showed that using UDP instead of TCP has the potential of reducing the latency of the MR application. However, this comes at the cost of higher bandwidth required for the communication link and the MR framework has to tolerate potential loss of frames. Thus, application developers or edge infrastructure providers will have to keep these trade-offs in mind when deploying their services.

Our study of the video compression format showed that our custom implementation of an MJPEG encoder/decoder for the downlink keeps the high frame rate of GStreamer using H.264 but at a lot lower latency. Further quality of experience (QoE) studies are however required in order to determine the impact of using a different video compression format on the quality of the resulting video depending on the bandwidth used. Indeed, the current prototype is not affected with regards to QoS by an unsufficient bandwidth on the downlink, but the end user QoE might be.

One further performance gain is to use hardware encoding instead of the software encoding used at the moment. Preliminary tests performed indicate that this has the potential of lowering the latency even more when using UDP.

5.6 Impact of Hardware Choices

FRTT is important in the studied MR application as it has a high impact on the QoS. If the FRTT is too high, the user will notice that the video stream containing the virtual elements is not matching what he/she is currently filming.

In this section we test two hypotheses. As visible in Table 5.1, although the bulk of the FRTT is spent for transmission, around 1% of the total (corresponding to 5% of the acceptable performance) is spent in queuing frames in the edge part since the MR framework is already busy. Therefore, the first hypothesis (H1) is that using a more powerful edge device will remove this queuing time, and thus reduce the FRTT. Then, as all the heavy computations are performed at the edge, the FRTT should not be impacted by changing the end device. This is the second hypothesis (H2).

5.6.1 H1: A better CPU improves FRTT

In order to test H1, we compare the performance of the prototype application when running on the baseline edge device and on the extra, high-end edge
device. Table 5.2 summarizes measurement statistics, where MR includes point cloud creation and graphic overlay creation.

In particular, we present in Figure 5.10a a breakdown of the average FRTT for different configurations. The results are surprising, because the average FRTT for the high-end configuration is 1.13 times higher than for the baseline configuration. Hence, an edge device with a more powerful CPU does not necessarily reduce the FRTT, it can actually increase it.

Even though the overall FRTT is increased, we zoomed in on the part of it that is spent at the edge. Figure 5.10b shows that the queuing time in the edge system is indeed reduced when using the high-end configuration. The MR framework is also executing faster. However, the high-end configuration had negligible impact on the time spent on graphics. We conjecture that to decrease this part, other improvements such as using graphic accelerators are needed, if the SLAM framework used can take advantage of it.

5.6.2 H2: FRTT is smartphone-independent

Chen et al. [27] showed that in the context of wearable devices, there is a performance impact based on the hardware used. With major computations done at the edge, we now compare the baseline device with an additional smartphone (Samsung), that, although a bit less powerful than the baseline one, should be fully capable of handling the MR application. The results are presented in Table 5.2.

Unexpectedly again, the values show that the Samsung configuration as 1.36 times higher FRTT on average. This indicates that some part of the application is handled differently by the two end devices considered. By looking at Figures 5.10a and 5.10b, it appears that it is the transmission part that behaves differently, i.e. either the encoding/decoding, the WiFi transmission or the way the TCP protocol is handled. Interestingly, preliminary results with UDP showed that the difference in FRTT in this case is lower between the two configurations, indicating that the protocol handling should be investigated further.

5.7 Summary and Possible Extensions

Mixed reality applications are a good candidate for offloading at the edge. Indeed, they require heavy computation and very low latency in order to deliver high quality of service to the end user. However, deploying a MR application that performs video streaming towards the edge and back is not trivial due to the different components that need to be brought together.

In this work, we found that bringing down the time spent in the communication link is critical for such MR applications and we see the different alternatives considered in this chapter as a first step towards further studies of this aspect.
5.7. Summary and Possible Extensions

A way forward is to investigate other possibilities for reducing the latency of the MR application such as using hardware acceleration for encoding/decoding, using other SLAM frameworks, using other types of protocols and dissecting combinations of those. For example, it would be interesting to investigate a custom protocol for MR that would be built on top of UDP but with some features of TCP such as guaranteed delivery and packet synchronization in order to keep the bandwidth needed low while avoiding frame loss. An example of an attempt to such protocol is ENet \(^7\), which is unfortunately not implemented in the video streaming frameworks available at the moment.

Moreover, it would be interesting to study the application with regards to its energy usage. Indeed, applications running mixed reality on the end device can drain its battery and it would be interesting to study how the offloaded mixed reality application performs in this regards.

Finally, this work could be extended to execute within a future 5G deployment where other protocols are in use and where hardware accelerators are exploited.

\(^7\)http://enet.bespin.org/
In this chapter, the method used for obtaining a workload model exhibiting the same characteristics as an application is presented. It is then applied on the MR-Leo prototype for obtaining a mixed reality workload model.

6.1 Method

When performing workload characterization and modeling, numerous indicators can be studied. The objective is that the model presented in this thesis is of relevance for researchers and developers of edge computing platforms and algorithms. Therefore, the characterization performed starts by determining what are the load indicators used in the state-of-the-art research. Then, a method is devised for extracting knowledge from the application with regards to those load indicators, and create a model of the studied workload.

6.1.1 Load Indicators

For a proposed workload model to be relevant, it needs to include the load indicators that will be considered by the model users, e.g. researchers within edge computing.

Load Indicators Used in Current Research

Our study is not the first one to look into workload characteristics that are of interest for edge computing research. Researchers working in benchmarking for edge computing have proposed characteristics to be taken into account when creating or using application workloads in their benchmarks.
To compare edge computing platforms, Das et al. [144] utilize workloads from three different applications that have different computation resource demands. When comparing the deployment of applications using different modes, McChesney et al. [117] use six applications that are characterized with one or several of the following type tags: latency critical, bandwidth intensive, location aware and computational intensive. Finally, the load indicators considered by Toczé et al. [111] in the perspective of benchmarking edge algorithms and techniques are the resource demand (relative to communication, computation and storage), the deadline, the arrival type (to be understood as arrival pattern) and the interarrival time.

In addition to the above, we also look at recent edge computing research evaluated using simulations. We selected 5 works published in 2018 or 2019 in relevant edge computing conferences where evaluations were based on a simulator. The simulators used were: iFogSim [145], EdgeCloudSim [116], SimGrid [146], and FogTorchPi [147]. They are all open-source and have been used by several groups of researchers.

Table 6.1 presents the outcome of this literature study. For each work, we checked whether the workload used considered the computation demand, the communication demand, the storage demand, the arrival pattern of tasks, some timing aspect (e.g. deadlines), the location of tasks or any other aspect. We focus on those as they are considered in the benchmarking efforts presented earlier.

As Table 6.1 shows, the computation and communication resource demand are indeed of high interest for edge computing research and it is therefore important to study applications with regards to those load indicators. Similarly the task arrival pattern and timing are important to study. The storage resource demand, although not considered by all works is still considered by some. When reviewing the literature, it could be noted that the location-related indicators come from the locality of the end devices (and its mobility, when relevant) and not from the task itself. Therefore, we will not consider this indicator further in this study.

Selected Load Indicators

Based on the above load indicators found in recent benchmarking research and in research conducted using simulators, we conclude that when characterizing an edge application, specific care should be taken considering:

- Task definition
- Task arrival pattern
- Resource demand (especially regarding computation and communication)
- Timing
Table 6.1: Load indicators considered in selected works using simulation.

| Article       | Computation | Communication | Storage | Arrival | Timing | Location | Other | Simulator tool |
|---------------|-------------|---------------|---------|---------|--------|----------|-------|----------------|
| De Maio [88]  | ✓           | ✓             |         |         |        | ✓        | ✓     | FogTorchPi     |
| Toczé [148]   | ✓           | ✓             |         | ✓       | ✓      | ✓        |       | EdgeCloudSim   |
| Sonmez [91]   | ✓           | ✓             |         | ✓       | ✓      |          | ✓     | EdgeCloudSim   |
| Xia [94]      | ✓           | ✓             | ✓       | ✓       | ✓      |          |       | SimGrid        |
| Mahmud [90]   | ✓           | ✓             | ✓       | ✓       |        |          |       | iFogSim        |
6. MODELING AN EDGE MIXED REALITY WORKLOAD

More specifically, during the characterization and modeling, it should become clear what an offloaded task is in the context of the application, and how often those tasks are offloaded to the edge. Then, it is important to characterize and model the resource demand of those tasks as well as their timing constraints, for example whether they have a deadline.

6.1.2 Method Overview

In this section, the method used for characterizing and modeling the mixed reality workload is described. It is presented in a generic way so that it could be reused for characterizing and modeling relevant and realistic edge workloads for different application types. An overview of the characterization and modeling workflow is presented in Figure 6.1.

The workflow is composed of three main steps (represented by rectangular boxes): 1) application understanding, 2) application instrumentation and 3) statistical analysis. Each of these steps will produce an artifact (represented by oval boxes). Those steps and artifacts will be described in more detail in the next section.

As an input, the workflow takes an application. It is considered that one has access to this application, i.e. it should be possible to review the application code and to instrument it. If one or both of these conditions are not satisfied, then one or both of the first two steps of the method cannot be performed. In that case, the method can still be used but the artifacts corresponding to the step(s) that are not possible will have to be provided by some external part (for example the company owning the software) that should be trustworthy. As an output from the workflow, a statistical model of the workload is obtained, that can be used as an input to a load generator.
6.1.3 Characterization and Modeling

The major part of the method introduced in Section 6.1.2 deals with characterizing the application, i.e. observing and documenting its behaviour. The concerned parts are depicted in dark gray in Figure 6.1. This is essential for our aim of generating realistic workloads because such workloads require some quantitative data to be suitable as input to a benchmark or a simulator. In addition to observing the behaviour of the application, i.e. characterizing it, we also want to model it in order to be able to synthetically generate a workload that will exhibit the same characteristics as the original workload. The modeling parts lead to the acquisition of the first and third artifacts presented on Figure 6.1 and are depicted in light gray.

Step 1: Application Understanding

The first step of the method consists in getting some deep understanding of the application. The input to this step is the application itself that will be analyzed by one expert in order to extract a qualitative understanding about the functioning of the application. With regards to the selected load indicators mentioned in Section 6.1.1, this step aims at defining what is an offloadable task in the context of this particular application. If an actual application is not available, this step can be performed at a theoretical level, however with the risk of not capturing all the different interactions that a real application will require.

Artifact 1: Application Model

The first artifact, the application model, is created based on the application understanding step. In this step, and based on the load indicators presented in Section 6.1.1, we want to define precisely what is an offloaded task in the context of this application. If relevant, other load indicators from the list can also be modelled at that stage. When an offloaded task is defined, the application can be modeled as a directed acyclic graph (DAG) that shows the flow of tasks within the application. In such a graph, the vertices are computational functions and the edges are communication links. Once the application model is created, i.e. what are the offloaded tasks and how they will be transmitted and computed are known, the second step that is application instrumentation can be performed as it will be clear what needs to be measured and where this should be done.

Step 2: Application Instrumentation

The second step of the method consists of instrumenting the application code in order to gather data about the selected load indicators presented in Section 6.1.1. This data consists of various measurements. The aim is to get
quantitative experimental data with regards to the suitable selected load indicators. At a general level, the focus should be on studying the tasks (as defined previously), how often they are created on the end device and how they are handled by the edge device, at which pace and using what amount of resources.

Artifact 2: Experimental Data
This artifact is the output of the application instrumentation, obtained by running experiments with the application. Depending on how the measurements are logged, some pre-processing of the data may be required before it can be used as an input for statistical analysis.

Step 3: Statistical Analysis
The last step of the proposed method, the statistical analysis is performed on the experimental data artifact. Using descriptive statistics and other statistical tools, the data can be visualized and analyzed in order to get quantitative data as an input to the workload model. Statistical tools such as similarity tests (e.g. Pearson’s test) can also be used to find reoccurring patterns in the experimental data, when relevant for the application according to the knowledge gained in the application understanding part.

Artifact 3: Statistical Workload Model
Based on the statistical analysis, information is obtained about how the load looks like. The purpose of creating the statistical workload model is to generate a synthetic workload that will be similar to the workload generated by the real application. The method that is used is to perform probability distribution fitting for the selected load indicators for which experimental data was gathered. For the load indicators were the gathering of experimental data was not possible, the knowledge from the application understanding step is used.

6.2 Task Definition
As part of the application understanding step, it should be defined what an offloadable task is in the context of our selected interactive MR application. To do that, the inputs and outputs to the application are analyzed.

In a typical use case scenario of the application, the user will capture some environment with her end device and then press a button on the screen to add a virtual element to the environment. This element can also be interactively removed, by pressing another button on the screen. The user moves the end device and can add another element at another place, and so on.
Therefore, the application has two types of input: video input (the video stream captured by the device camera) and user input (the user pressing a button to add or remove a virtual element). However, it only has one output: the resulting video stream that contains the original stream enhanced with the virtual elements when appropriate.

These two types of input have very different characteristics and should therefore be considered as two different types of task that the server part of the application will have to handle. We look deeper at those two to define the task types for MR-Leo.

Regarding the video input, each video frame is sent to the MR function on the edge device, that will perform analysis of the frame. For obtaining good quality of service, each frame should be analyzed and result in an output frame displayed to the end user. This output may or may not include virtual elements. Therefore, it is natural to define a task for the video input as the handling of one frame. This first task type will be referred to as video task in the rest of the chapter.

Regarding the user input, each such input should be handled by the MR function at the edge, before the MR service is provided to the end user. Therefore, handling the user input is also defined as a task. This second task type will be referred to as a user task.

6.3 Task Arrival Pattern

For each identified task that will be offloaded to the edge we need to characterize the task arrival pattern, to be able to later generate the synthetic workload. The task arrival pattern corresponds to how often a task will have to be handled at the edge and is obtained as a result of applying both steps 1 and 2 of the method.

When defining where different measurements points could be placed, it was considered that the communication link was perfect. Indeed, modeling the communication link is out of the scope of this work. This is particularly relevant when considering the task arrival pattern. Theoretically, the task arrival pattern should be the same as the task generation pattern at the end device. However, experimental data gathered by logging the video task in terms of the number of frames received per second at the edge (presented in Figure 6.2 for 30 experimental runs) shows that is it not the case in practice. Although the video frames are generated at a fixed rates of 30 fps, those same frames arrive at the edge with a frame rate varying between 20 and 32 when considering the first 59 seconds (i.e. 98%) of the video. This is due to the choice of the video encoding and transport protocol adopted for the implementation. Those choices are, however, not related to how the workload of the application looks like in the generic case. For the sake of edge-side appli-
6. Modeling an Edge Mixed Reality Workload

![Experimental frame arrival rate at the edge device.](image)

Figure 6.2: Experimental frame arrival rate at the edge device.

In our simulation modelling we will ignore the transport and coding-induced jitter and will model the task arrival pattern based on the task generation pattern\(^1\).

Starting with the video tasks, the task generation pattern is the same as the frame generation pattern, per definition. Since the video considered for the MR-Leo implementation is shot at 30fps, the task generation rate is 30 tasks per second and the interarrival time is constant. Therefore, the video tasks should be generated periodically with an interarrival time of 33ms.

For the user tasks, the task generation pattern depends on how often the user will press the button on the screen. Based on some observations during the testing of the prototype, it appears that a typical user presses the button to add/remove a virtual object a few times per minute. For user tasks, the arrival can thus be modelled as a Poisson process with an intensity of \(\lambda = 5\) tasks per minute. This will be a parameter that one can change upon configuring the model differently.

Note that this will mean that there are 1800 video tasks generated per minute in parallel with around 5 user tasks. Hence, a model similar to earlier works [91] where a Poisson process is used to create task arrival patterns is not realistic. Such a model will miss to represent the periodic arrival of video tasks that represent 99% of all tasks.

\(^1\)Modelling additional variations can be added at a later stage if needed.
6.4 Resource Demand

In this section, we describe how the resource demand part of the workload model was obtained according to the method. First, we detail how we applied steps 2 and 3 as this is relevant for understanding the results obtained e.g for the computation demand. We mainly focus on the computation demand, which is the most common resource considered in the literature study presented in Section 6.1.1. We also consider the communication and memory demand, which are the next two significant resources.

6.4.1 Overview of Applying Steps 2 and 3

In this case study, the application instrumenting step reused part of the instrumentation done for evaluating the performance evaluation of MR-Leo presented in Chapter 5, specifically logging the task arrival rate on the edge device and on the end device. In addition, the application was instrumented in order to log the CPU use and the memory use at every second. This was done using the Linux `top` tool. In order to get the number of executed instructions, a separate instrumentation of the application code was performed using a custom code building on the Intel Pin tool\(^2\) for binary code analysis.

To obtain data, the experiments were run on a HP Elitebook 840 G5 running Ubuntu 18.10 with 16 GB RAM and an Intel Core i7-8550U CPU. Two sets of experiments were run: first 30 runs using a pre-recorded video as in Chapter 5 (later denoted as the reference video) in order to log the task arrival as well as CPU and memory use. Automatic user input were added in order to also include this workflow in a controlled way. Then, a separate set of experiments were conducted for getting the number of instructions executed per video frame. Separate runs were conducted for video input alone, and for video and user input combined. The reason why this data was gathered in separate runs is that binary code analysis slows down significantly the execution of the application and would therefore influence the other metrics. Due to this slow down, this test set had to be performed with a mocked smartphone client, i.e. the video frames are directly used as input on the edge server.

For the statistical analysis, the following techniques are used: first, the data obtained using histograms are plotted in order to visualize how the data behaves. In addition, descriptive statistics such as the mean, median, maximum and minimum values are used for further analysis of the data. The presence of one or several modes in the data is also studied. The mode of a set of values is the value appearing most often [149]. It appears as a local maximum in the probability density function of the data distribution. In some distributions, there are several such local maxima. In this case the data is multimodal and the different modes can be identified by statistical tools.

\(^2\)https://software.intel.com/en-us/articles/pin-a-binary-instrumentation-tool-downloads
The experimental data gathered is then used to derive numerical data for the statistical workload model. In particular, for the computation demand, a bimodal normal distribution is fitted to the data using the R package mixtools. The fit of the obtained distribution is checked using the Kolmogorov-Smirnov (KS) tests and Pearson’s chi-squared test, two statistical tests for studying different similarity indicators that are suitable for comparing two distributions. The KS test measures the maximum difference between two cumulative distribution functions and its result can be seen as describing the divergence of the two distributions [121]. The Pearson’s chi-squared test measures the difference between the histograms of two empirical distributions [121]. The fit of the obtained distribution is visualized by plotting the quantile-quantile (QQ) plot between the experimental data points and a set of data points generated from the obtained distribution. In such a plot, the points are aligned along the line $y=x$ if the two distributions are similar.

### 6.4.2 Computation Demand

Characterizing the computation demand is not a straightforward task. Based on the literature study performed in Section 6.1.1, the computation demand is sometimes not quantified but described in generic categories such as “High” or “Low” [117, 111]. When it is quantified, the metric chosen varies and the numbers are provided without explaining how they were obtained. In this study, we use two metrics: the number of cores used and the number of instructions executed during running a task.

#### Number of Cores

For characterizing the CPU load of the MR application, the CPU use during running the edge processing part of the application was logged on the edge device. The result is that the MR-Leo application requires the power of three cores. This can be seen on Figure 6.3, showing a histogram of the CPU usage values recorded during 30 experimental runs. On the figure, the maximum CPU utilization goes up to 226%, i.e. the application requires the equivalent of two cores utilized at 100% and a third one utilized at 26%, so three (logical) cores are needed.

Therefore, if the computational demand is modeled as CPU cores, then the computational demand deriving from the method is 3 high-end cores\(^3\).

#### Number of Instructions per Video Task

With regards to the number of instructions run and as described in Section 6.4.1, a dynamic binary analysis is performed with a custom adapted tool based on Pin. This tool enables to count the number of instructions executed

\(^3\)I.e. equivalent to the Intel Core i7-8550U logical CPU cores
6.4. Resource Demand

![Figure 6.3: Measured CPU use per second.](image)

|                  | Avg | $\sigma$ | Median | Min   | Max   |
|------------------|-----|----------|--------|-------|-------|
| Reference video  | 244.08 | 98.53  | 185.30 | 119.48 | 563.91 |
| Object video     | 185.41 | 43.98  | 164.03 | 133.03 | 316.02 |
| Campus video     | 269.41 | 42.43  | 282.42 | 140.52 | 384.54 |

Table 6.2: Basic statistics for the instruction count (MI).

between two points in the source code. We use this tool to count the number of instructions executed when processing a frame in the class that feeds the decoded frames to the MR function.

The analysis was run 5 times on the video stream captured for the performance evaluation presented in Chapter 5. Figure 6.4 shows a histogram of the number of instructions per frame over the 5 runs. Basic statistics over the 5 runs, are presented in Table 6.2. On the histogram, it can be seen that the numbers of instructions are not spread homogeneously over the span between the minimum and the maximum values. On the contrary, the data exhibits two modes: one at 158.43 million instructions (MI) and a second one at 310.03 MI.

When plotting the data for each run separately, the shape of the histogram is similar. In order to quantify this similarity, we compare the output of each run to the other ones using the Pearson’s chi-squared test. Twelve bins were considered for grouping the data. The tests indicate similarity between the distributions obtained from the different runs (with p-values between 0.23 and 0.24).
As mentioned earlier, the data presented two modes. For modeling it, a distribution fitting was performed using the R library mixtools. This assumes that the distribution is a mix of two normal distributions, which seemed reasonable having seen the shape of the histogram presented in Figure 6.4. The density function of the obtained distribution is presented in Figure 6.5. The two distributions composing it are a normal distribution with mean $\mu = 155.62$ MI and standard deviation $\sigma = 14.10$ for 47% of the values and a normal distribution with mean $\mu = 322.38$ MI and standard deviation $\sigma = 71.18$ for the remaining 53%.

The fit is tested using a QQ plot shown on Figure 6.6 and by using the Pearson’s chi-squared test and the KS test between sampled data from the fitted distribution and the experimental data. All show a good fit. The p-value for the Pearson’s chi-squared test was 0.23 and the output of the KS test was 0.026.

We conjecture that the two modes of the distribution, i.e. the two levels of computational intensity, correspond to two situations in which the application can be: 1) it needs to create a point cloud, and 2) it needs to update it. The first situation being more computationally intensive than the second one. In order to test this, two additional series of tests were conducted with two other videos. The first one shows different objects disposed on a table (referred below as the object video) while the second one is shot in an outside environment.

---

4Available at https://gitlab.liu.se/ida-rtstab/public-code/2019_mrleo_charac
6.4. Resource Demand

Figure 6.5: Density function of the fitted distribution.

Figure 6.6: QQ-plot of the data vs the fitted distribution.
6. Modeling an Edge Mixed Reality Workload

![Figure 6.7: Measured number of instructions per frame.](image)

environment on the campus of Linköping University (referred below as the campus video). The two additional videos contain the same number of frames as the reference video. When using the MR-Leo implementation, those two videos exhibit different behaviours compared to each other and to the reference video used previously. For the object video, a point cloud can be found by the MR-Leo application and once found, is kept until the end of the video. On the contrary, a point cloud can never be found for the campus video, as the scene is too complex for the MR framework used (although it reflects the state-of-the-art). The reference video exhibits a behaviour in between those two extremes, with a point cloud being created but lost for a while around 30 seconds, and then recovered.

When plotting the histogram of 5 runs using those videos, it can be seen that the number of instructions executed is different in the three different scenarios proposed by the three videos. Figure 6.7a shows the histogram for the campus video. This histogram is unimodal with a mode at 287.46 MI. Figure 6.7b shows the histogram for the object video. This histogram is actually trimodal, with two modes close to each other (160.45 MI and 172.24 MI) that are aggregated in the first bin of the histogram, and a third one at 269.59 MI, containing only 13% of the data values. Basics statistics for the campus and object videos can be found in Table 6.2. The results of those experiments therefore seems to validate the conjecture made as the campus video is always trying to create a point cloud, without success, while the object video will mostly update the point cloud once it is created.

**Number of Instructions per User Task**

The next aspect we study is the difference in terms of instructions executed depending on whether a user task has been sent to the edge node or not. The experiment performed with the reference video is adapted to include an automatic trigger of a user task adding a virtual element during 100 frames,
6.4. Resource Demand

(a) With a VE

(b) Without a VE

Figure 6.8: Measured number of instructions per frame.

and then removing the virtual element during the next 100 frames, and so on during the whole experiment length. This adapted experiment is run 5 times.

We plot the histogram of the frames including a virtual element on Figure 6.8a and of the frames not including a virtual element on Figure 6.8b. The two distributions are compared using the KS test and Pearson’s chi-squared test. The KS output is 0.028 and the p-value obtained is 0.086 for the Pearson’s chi-squared test. This indicates that the two distributions are not significantly different.

Based on the above tests and the fact that user tasks only account for 1% of all tasks, we do not include a separate model for the computation demand of a user task. Instead the model for computation demand of video tasks is deemed sufficient for frames including a virtual element and frames not including a virtual element.

6.4.3 Communication Demand

As part of the application understanding part, it needs to be highlighted in this study that the MR-Leo application will have an almost symmetric amount of data transmitted for upload and download, contrary to what is sometimes assumed for MR applications (e.g. in [91]). This is because all the MR parts, including rendering, are offloaded to the edge and it is needed to transmit a full frame to the edge, and also a full frame back to the end device. The only part that is not symmetric is the one corresponding to user tasks, as that are only transmitted from the end device to the edge device. Those tasks however represents only a tiny fractions of all tasks as described earlier.

Regarding video tasks, we study how they look like when generated on the end device. The video stream used for performing the performance evaluation presented in Chapter 5 is taken as an example of a standard input for the MR-Leo application. This video stream includes 1814 frames for a size of
6. Modeling an Edge Mixed Reality Workload

Figure 6.9: Histogram of the memory usage of the MR application.

75.5 MB, which corresponds to each frame having a size of around 41 kB before encoding. How much data is going to be actually transmitted between the end device and the edge device will depend on the video compression standard chosen. Modelling the behaviour of different compression standards (such as H.264 or MJPEG that are included in MR-Leo) is relevant but out of the scope of the work presented in this chapter.

For user tasks, the size of a request can be obtained by studying the implementation. It is set as 60 bytes: 10 bytes for the header and 50 data bytes. The user tasks are not encoded so their size is independent of the compression format used.

6.4.4 Memory Demand

When studying the application during the application understanding step, one outcome is that the MR-Leo application does not store data permanently on the edge server but only requires memory during run time. This was measured using the same instrumentation as for the CPU use.

The memory usage of the MR-Leo application was thus measured each second during 32 experimental runs. Figure 6.9 shows a histogram of the measurements, which include the memory used by the application before the end device connects, during the time it is connected and after the end device disconnection.

Two levels of memory occupation clearly appear on Figure 6.9. The first one, around 1 281 kB corresponds to the memory usage of the application when no end device is connected. The second, at on average 3 735 kB (with a standard deviation of 82 kB) corresponds to the memory usage when an end
device is connected, i.e. to the size of the application and the environment model associated to the connected end device. During the connection time of the end device, the memory occupied usually increases and decreases of a few tens of kB, but without any obvious pattern related to the video stream used for the test.

The model proposed for the memory demand of the considered MR application is thus as follows: an individual task (video or user) does not have a specific storage requirement but the edge device needs to provide 1.3 MB of memory and an additional 2.4 MB per end device connected to this edge device.

### 6.5 Timing

The timing aspect includes defining a task deadline and expressing the latency-sensitivity of the application. This is therefore part of the first step, application understanding.

As any application relying on live video streaming, the considered MR application is very sensitive to jitter. Getting video frames late and at an unsteady pace will degrade the quality of experience of the user. Therefore, when using an MR workload, it will be important to measure how many tasks met their deadline to determine whether the solution meets the per task delay requirements inherent to such an application.

In order to determine a suitable deadline for the two types of tasks, we use the performance evaluation of the MR-Leo implementation that was presented earlier in Chapter 5. In this study, it was observed that using some hardware configuration for an edge server will lead to video frames being queued because the edge device was unable to process one frame before the next one arrived, due to a lack of computational resources. If this happens too often, it leads to a degradation of the quality of service of the application, as it displays outdated frames to the user. The implementation therefore includes a mechanism for discarding old frames but this comes at the cost of a decreased frame rate at the end device.

Therefore, in order to avoid both outdated frames and discarded frames, it is desirable to avoid frames queuing up at the edge end. This will improve the quality of service of the application both with regards to time for handling a frame (that will increase with queuing) and to the number of tasks completed (that will decrease if frames are dropped). Therefore, an ideal edge system should process completely one frame before the next one comes. Hence since the frame rate of the prototype is 30 fps, the video task deadline can be set to 33ms. Missing this deadline will not prevent the application to work but will degrade the quality of service.

Regarding the user task deadline, the graphics will be added or removed in the next coming video task. Therefore the deadline for such a task is at
least twice the deadline for a video task, i.e. 66 ms. This is under 100 ms, which is the limit considered for perceiving the response as immediate [142].

6.6 Mixed Reality Prototype Model

As an output from the method presented in Section 6.1, two artifacts are obtained: an application model and a statistical workload model.

6.6.1 Application Model

Based on the application understanding step, an application model artifact is created. As described in Section 6.1.3, the model is a DAG where vertices are computational functions and the edges communication links. Some simulators, such as iFogSim, require the application to be modeled in this way.

The application model for MR-Leo is shown in Figure 6.10. It includes computational functions related to the two different inputs discussed earlier (the user input and the video input), as well as to the output of the application, which is displayed to the end user. Those three functions are the only ones executing on the user device, which is depicted with a light gray background. On the edge device, two other functions are executed. Those are the offloaded functions depicted with a dark gray background. The first and main one is the Mixed Reality function, where all the computations related to providing the MR service are performed. The second one, called Communication service is in charge of receiving, queuing and sending the inputs to the MR functions and sending the output to the end device.

It should be noted that this application model can be used for any interactive MR application offloading all the MR computations to the edge, not only MR-Leo. Indeed, all implementation specifics are abstracted away in this representation, such as the choice of encoders or content of the MR framework used. However, we used our knowledge about the architecture of a known application to provide the abstraction that we believe is implementation-independent.

6.6.2 Statistical Workload Model

The proposed statistical workload model for fully-offloaded MR applications based on applying the method presented in Section 6.1 to the MR-Leo implementation is summarized in Table 6.3. This is the third artifact created by the method and it can be used as an input to simulators and other types of load generators.
6.6. Mixed Reality Prototype Model

![Interactive MR application model diagram]

Figure 6.10: Interactive MR application model.

| Aspect       | Model                                                                 |
|--------------|----------------------------------------------------------------------|
| Video task   |                                                                      |
| Task arrival | Periodic with interarrival time 33 ms                                |
| Computation  | For 47%: $\mathcal{N}(\mu = 155.62, \sigma = 14.10)$               |
|              | For 53%: $\mathcal{N}(\mu = 322.38, \sigma = 71.18)$               |
| Communication| 41 kB (full frame, upload and download)                              |
| Deadline     | 33 ms                                                                |
| User task    |                                                                      |
| Task arrival | Poisson distributed with $\lambda = 5$ (min)                         |
| Communication| 60B                                                                  |
| Deadline     | 66 ms                                                                |
| App Memory   | Application alone: 1.3 MB                                            |
|              | Per end device connected: add 2.4 MB                                 |

Table 6.3: Overview of the proposed MR workload model.
6. MODELING AN EDGE MIXED REALITY WORKLOAD

6.7 Discussion

In this section, we first compare the proposed workload model to other MR models found in the literature and then present limitations of this work.

6.7.1 Comparison to Existing Model

The workload model presented in Section 6.6.2 present several differences with other models previously presented in the literature for MR applications. The main one is that it is based on a real implementation. We illustrate the added value of the proposed model by comparing it to one of the currently existing models that considered similar load indicators: the AR application model with numerical data used by Sonmez et al. [91] in their evaluation.

With regards to the first selected load indicator of Section 6.1.1, task arrival pattern, the current model considers exponentially distributed task interarrival time (with $\lambda = 2$ seconds) combined to the concept of active and idle periods. This means that the application alternates a phase where no tasks are generated (the idle period, of length 20 seconds) with a phase where tasks are generated with a mean interarrival time of 2 seconds (the active period, of length 40 seconds). Therefore, the load generated with such a model will have a very different profile compared to the MR-Leo load in terms of how many and how often tasks arrive to the edge node. This is illustrated in Figure 6.11, that shows an example of the arrival times of tasks during 3 seconds when using the two models for generating tasks. The number of tasks is more than 8 times higher using the model based on MR-Leo (90 tasks vs 11), which can strain the resources more. Therefore, testing a task placement algorithms with a model like the current one may lead to unreliable conclusions as the load used is not going to be representative for the real application load.

With regards to the timing aspect, the model from [91] only include a delay sensitivity parameter for a task that indicates if the task is delay-tolerant or not but no deadline. As discussed earlier, avoiding frame queuing is crit-
6.7. Discussion

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.

Figure 6.12: Density function of the number of instructions per task in the Sonmez model.
6. Modeling an Edge Mixed Reality Workload

6.7.2 Relevance and Validity

One obvious concern with this work may be that the numerical data used for the characterization and as an input to the MR application model is coming from a specific MR implementation, when used in a specific configuration (H.264 video transmitted over TCP), and using some specific hardware. This MR implementation is of type fully-offloaded, i.e. it offloads all the MR computation at the edge. Therefore, the numerical data obtained may not be the same when considering another MR application type (such as one performing some rendering on the end device), another algorithm, another configuration or other hardware. However, we argue 1) that the fully-offloaded type is relevant as it is representative of the type of MR algorithms with full offloading in order to run on a great range of end devices, 2) the MR-Leo is to the best of our knowledge the only available open-source implementation for such an application type, and 3) the combination of H.264 over TCP is widely used for video streaming and 4) the hardware used is currently reasonable to use in an edge configuration. Moreover, the method for characterization and modeling is applicable to a wide range of applications so that similar models can be derived from other applications. The MR-Leo implementation is released open-source so it is also possible to modify this particular application to run different configurations or with different hardware to extend this study.

A second concern may be that the numerical data was obtained using a limited set of pre-recorded videos. Thus, it is possible that the application will perform differently when getting another type of input. However, the standard video stream was shot to include some challenging parts for the MR framework used (e.g. it moves out of the original scene and then comes back again). This has been confirmed during the performance analysis presented in Chapter 5 so we believe that the measurements are representative for a good enough combination of situations where the computations exhibit different complexity levels. Moreover, with the prototype being available open-source, it is very easy to extend this study with experiments performed with other types of video input.
This final chapter presents the conclusions based on the thesis work and discusses interesting directions for future research.

7.1 Conclusions

In this thesis, we have focused on the resource management aspect of edge computing and studied the state of the art of the research conducted in this broad and emerging area in order to identify which parts are well studied and which parts are still missing.

When diving into the supply side, the edge computing paradigm enables new possibilities for providing resources, where one of them is to have mobile edge devices. As seen in Chapter 3, this is a still largely unexploited field that is promising for enabling ultra low latency for the already mobile end devices. When studying such a scenario, it was important and challenging to both place the different tasks that have to be executed, but also to define a strategy for how the edge devices should move.

When considering the resource demand side, we studied in depth one of the typical edge applications (mixed reality) in order to learn more about its resource demand. As such applications are still not wide-spread, we had to develop our own research prototype in order to perform our investigations. With the insights and measurements obtained, we built a mixed workload model that exhibits the same characteristics as the MR-Leo application.

The taxonomy proposed in Chapter 3 and the associated review of the state of the art shows that most of the current research about resource management for edge computing is focused on a few aspects, including for example service placement and resource optimization. While those aspects are important and should be dealt with, it is also crucial to explore some less
7. CONCLUSIONS AND FUTURE WORK

researched aspects that are new to the paradigm and promising. Those include the mobility of edge nodes, distributed organizations for the resource supply, and security for edge computing.

Our high level simulations of a coarse-grained representation of the resource supply organization show that mixing mobile edge nodes with stationary ones is promising for providing good quality of services for tasks that require ultra low response time. This first evaluation performed in a simulated environment has to be extended to evaluate the applicability of this innovative resource supply organization, its performance, and possible assets compared to more traditional ways of organizing resource supply. This could be done through the realization of a small scale proof-of-concept implementation of the ORCH framework.

With regards to resource demand, the lack of edge applications available open-source is restricting the quantity and diversity of realistic workloads that are available to use for edge computing studies. By providing an open-source mixed reality prototype as well as a study of its characteristics and a stochastic model of the workloads it creates, the thesis contributes to filling the gap for one of the most cited use cases of edge computing. Similar work has to be conducted with other applications in order to get a broad set of realistic edge workloads that can be used for edge computing research.

7.2 Future Work

Resource management for edge computing is a recent but already very active area of research. As shown in Chapter 3, it is very broad and our work focused only on some parts of the area, although there are a lot of possible further directions of research. Several of them are listed here.

In the thesis, only one specific application from one application domain was characterized and its workload modeled. In order to enable studies on the broad range of envisioned edge applications, more applications have to be created, made available, studied and modeled. As a motivation to gather those workloads, a study of the added value of using realistic workloads instead of the currently used ones should be conducted.

In the ORCH framework instance presented in the thesis, the alternatives for placing a task are limited to the closest mobile edge device and the closest stationary edge device. It would be interesting to expand the set of devices considered to also include other neighboring edge devices as possible candidate for offloading. Exploring this direction requires to define criteria for the selection of offloading targets in that context.

Further continuation of the work on ORCH should include an actual deployment of it on a physical testbed. We believe that this would strengthen the work by taking into account some cost or overhead that can be too easily ignored in high-level simulation.
7.2. Future Work

The algorithms presented in Chapter 4 as part of a first proposal for the ORCH framework are straightforward but effective ones. We believe that it is needed to study more advanced algorithms for the different ORCH components in order to improve the quality of service. Some algorithms that would be especially interesting to study are adaptive algorithms or models, as those could account for dynamic changes in the environment of the system.

Offloading an application to the edge or managing edge devices does not come without a cost. A further direction that is interesting is to study the costs of doing so for both the end user and the edge owner. Quantifying and modeling them would enable a better understanding of the edge systems and of the possible bottlenecks for a wide adoption of the edge paradigm.

One of the identified resources that has not yet been considered is our work is energy. In-depth studies of the energy aspect of offloading at the edge as well as of mobile edge devices that will require energy for their mobility are needed. For this, we see that new tools have to be created in order to facilitate such studies in the specific context of edge computing.

Last but not least, edge computing can only be successfully deployed if the edge infrastructure is secured, i.e. it is crucial to prevent attackers to either get access to the data it contains or to disrupt the edge services. Indeed, if the edge is to be used for critical applications such as connected vehicles or health-care, it needs to have a high level of security. This is not possible to achieve without dedicating resources to it and it is thus an interesting direction for future work.
Bibliography

[1] J. R. Evaristo, K. C. Desouza, and K. Hollister, “Centralization momentum: The pendulum swings back again,” *Communications of the ACM*, vol. 48, no. 2, pp. 66–71, February 2005.

[2] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, “Edge analytics in the internet of things,” *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, April 2015.

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

[4] Ericsson. (2019, June) Ericsson mobility report. [Online]. Available: https://www.ericsson.com/49d1d9/assets/local/mobility-report/documents/2019/ericsson-mobility-report-june-2019.pdf

[5] N. Fernando, S. W. Loke, and W. Rahayu, “Mobile cloud computing: A survey,” *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, January 2013.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (MCC ’12)*, August 2012, pp. 13–16.

[7] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, January 2017.
[8] R. Roman, J. Lopez, and M. Mambo, “Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges,” *Future Generation Computer Systems*, vol. 78, no. 2, pp. 680 – 698, January 2018.

[9] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, and E. de Lara, “Cloudpath: A multi-tier cloud computing framework,” in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC ’17)*, October 2017, pp. 20:1–20:13.

[10] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, and Y. Zhang, “Mobile edge cloud system: Architectures, challenges, and approaches,” *IEEE Systems Journal*, pp. 2495–2508, September 2017.

[11] A. Mehta, W. Tärneberg, C. Klein, J. Tordsson, M. Kihl, and E. Elmroth, “How beneficial are intermediate layer data centers in mobile edge networks?” in *Proceedings of the 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, September 2016, pp. 222–229.

[12] T. Taleb and A. Ksentini, “Follow me cloud: interworking federated clouds and distributed mobile networks,” *IEEE Network*, vol. 27, no. 5, pp. 12–19, September 2013.

[13] A. S. Gomes, B. Sousa, D. Palma, V. Fonseca, Z. Zhao, E. Monteiro, T. Braun, P. Simoes, and L. Cordeiro, “Edge caching with mobility prediction in virtualized LTE mobile networks,” *Future Generation Computer Systems*, vol. 70, pp. 148 – 162, May 2017.

[14] V. Xhagjika, L. Navarro, and V. Vlassov, “Enhancing real-time applications by means of multi-tier cloud federations,” in *Proceedings of the 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, November 2015, pp. 397–404.

[15] F. Lobillo, Z. Becvar, M. A. Puente, P. Mach, F. L. Presti, F. Gambetti, M. Goldhamer, J. Vidal, A. K. Widiawan, and E. Calvanese, “An architecture for mobile computation offloading on cloud-enabled LTE small cells,” in *Proceedings of the 2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, April 2014, pp. 1–6.

[16] K. Wang, M. Shen, J. Cho, A. Banerjee, J. Van der Merwe, and K. Webb, “MobiScud: A fast moving personal cloud in the mobile network,” in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges (AllThingsCellular ’15)*, August 2015, pp. 19–24.

[17] J. Liu, T. Zhao, S. Zhou, Y. Cheng, and Z. Niu, “CONCERT: a cloud-based architecture for next-generation cellular systems,” *IEEE Wireless Communications*, vol. 21, no. 6, pp. 14–22, December 2014.
[18] P. T. Endo, A. V. de Almeida Palhares, N. N. Pereira, G. E. Goncalves, D. Sadok, J. Kelner, B. Melander, and J. E. Mangs, “Resource allocation for distributed cloud: concepts and research challenges,” *IEEE Network*, vol. 25, no. 4, pp. 42–46, July 2011.

[19] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, “Femto clouds: Leveraging mobile devices to provide cloud service at the edge,” in *Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 9–16.

[20] K. Habak, E. W. Zegura, M. Ammar, and K. A. Harras, “Workload management for dynamic mobile device clusters in edge femtoclouds,” in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC ’17)*, October 2017, pp. 6:1–6:14.

[21] M. Chiang and T. Zhang, “Fog and IoT: An overview of research opportunities,” *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, December 2016.

[22] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Mobile edge computing: A survey,” *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, February 2018.

[23] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, “Fog computing for sustainable smart cities: A survey,” *ACM Computing Surveys*, vol. 50, no. 3, pp. 32:1–32:43, June 2017.

[24] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, “Mobile augmented reality survey: From where we are to where we go,” *IEEE Access*, vol. 5, pp. 6917–6950, April 2017.

[25] A. C. Baktir, A. Ozgovde, and C. Ersoy, “How can edge computing benefit from software-defined networking: A survey, use cases, and future directions,” *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2359–2391, June 2017.

[26] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, “Integrated blockchain and edge computing systems: A survey, some research issues and challenges,” *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1508–1532, January 2019.

[27] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, and M. Satyanarayanan, “An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance,” in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC ’17)*, October 2017, pp. 14:1–14:14.
[28] R. Cziva and D. P. Pezaros, “Container network functions: Bringing NFV to the network edge,” *IEEE Communications Magazine*, vol. 55, no. 6, pp. 24–31, June 2017.

[29] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: The communication perspective,” *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, August 2017.

[30] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, March 2017.

[31] ETSI. (2019) ETSI GS MEC 003 V2.1.1 - Multi-access Edge Computing (MEC); framework and reference architecture. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01/gs_MEC003v020101p.pdf

[32] IEEE Standards Association. (2018) IEEE 1934-2018 - IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing. [Online]. Available: https://standards.ieee.org/standard/1934-2018.html

[33] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, “A comprehensive survey on fog computing: State-of-the-art and research challenges,” *IEEE Communications Surveys Tutorials*, pp. 416–464, November 2017.

[34] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, “Fog computing: Survey of trends, architectures, requirements, and research directions,” *IEEE Access*, vol. 6, pp. 47 980–48 009, August 2018.

[35] D. Henze, P. Schmiedmayer, and B. Bruegge, “Fog horizons – a theoretical concept to enable dynamic fog architectures,” in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing (UCC’19)*, December 2019, pp. 41–50.

[36] A. Mtibaa, K. A. Harras, K. Habak, M. Ammar, and E. W. Zegura, “Towards mobile opportunistic computing,” in *Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 1111–1114.

[37] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for VM-based cloudlets in mobile computing,” *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, October 2009.
[38] H. Frank, W. F. Fuhrmann, and B. V. Ghita, “Mobile edge computing: Requirements for powerful mobile near real-time applications,” in Proceedings of the Eleventh International Network Conference (INC 2016), July 2016, pp. 63–66.

[39] M. Aazam, M. St-Hilaire, C. H. Lung, and I. Lambadaris, “PRE-Fog: IoT trace based probabilistic resource estimation at fog,” in Proceedings of the 2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC), January 2016, pp. 12–17.

[40] A. Singh, N. Auluck, O. Rana, A. Jones, and S. Nepal, “RT-SANE: Real time security aware scheduling on the network edge,” in Proceedings of the 10th International Conference on Utility and Cloud Computing (UCC ’17), December 2017, pp. 131–140.

[41] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, “Greening the internet with nano data centers,” in Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT ’09), December 2009, pp. 37–48.

[42] H. Meng, K. Zheng, P. Chatzimisios, H. Zhao, and L. Ma, “A utility-based resource allocation scheme in cloud-assisted vehicular network architecture,” in Proceedings of the 2015 IEEE International Conference on Communication Workshop (ICCW), June 2015, pp. 1833–1838.

[43] M. Agiwal, A. Roy, and N. Saxena, “Next generation 5G wireless networks: A comprehensive survey,” IEEE Communications Surveys Tutorials, vol. 18, no. 3, pp. 1617–1655, February 2016.

[44] F. Z. Yousaf and T. Taleb, “Fine-grained resource-aware virtual network function management for 5G carrier cloud,” IEEE Network, vol. 30, no. 2, pp. 110–115, March 2016.

[45] K. Wang, K. Yang, X. Wang, and C. S. Magurawalage, “Cost-effective resource allocation in C-RAN with mobile cloud,” in Proceedings of the 2016 IEEE International Conference on Communications (ICC), May 2016, pp. 1–6.

[46] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, “Towards QoS-aware fog service placement,” in Proceedings of the 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), May 2017, pp. 89–96.

[47] H. R. Arkian, R. E. Atani, and A. Pourkhalili, “A cluster-based vehicular cloud architecture with learning-based resource management,” in Proceedings of the 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, December 2014, pp. 162–167.
[48] T. Nishio, R. Shinkuma, T. Takahashi, and N. B. Mandayam, “Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud,” in Proceedings of the First International Workshop on Mobile Cloud Computing & Networking (MobileCloud ’13), July 2013, pp. 19–26.

[49] D. Mascitti, M. Conti, A. Passarella, and L. Ricci, “Service provisioning through opportunistic computing in mobile clouds,” Procedia Computer Science, Special Issue Fourth International Conference on Selected Topics in Mobile and Wireless Networking (MoWNet’2014), vol. 40, pp. 143–150, September 2014.

[50] W. Liu, T. Nishio, R. Shinkuma, and T. Takahashi, “Adaptive resource discovery in mobile cloud computing,” Computer Communications, vol. 50, pp. 119–129, September 2014.

[51] W. Liu, R. Shinkuma, and T. Takahashi, “Opportunistic resource sharing in mobile cloud computing: The single-copy case,” in Proceedings of the 16th Asia-Pacific Network Operations and Management Symposium, September 2014, pp. 1–6.

[52] T. Penner, A. Johnson, B. V. Slyke, M. Guirguis, and Q. Gu, “Transient clouds: Assignment and collaborative execution of tasks on mobile devices,” in Proceedings of the 2014 IEEE Global Communications Conference, December 2014, pp. 2801–2806.

[53] M. K. Bekele, R. Pierdicca, E. Frontoni, E. S. Malinverni, and J. Gain, “A survey of augmented, virtual, and mixed reality for cultural heritage,” Journal on Computing and Cultural Heritage, vol. 11, no. 2, pp. 7:1–7:36, Mar. 2018.

[54] P. Milgram and F. Kishino, “A taxonomy of mixed reality visual displays,” IEICE Transactions on Information and Systems, vol. 77, no. 12, pp. 1321–1329, December 1994.

[55] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, “Visual simultaneous localization and mapping: A survey,” Artificial Intelligence Review, vol. 43, no. 1, pp. 55–81, January 2015.

[56] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, “Simultaneous localization and mapping: A survey of current trends in autonomous driving,” IEEE Transactions on Intelligent Vehicles, vol. 2, no. 3, pp. 194–220, September 2017.

[57] S. Yi, C. Li, and Q. Li, “A survey of fog computing: Concepts, applications and issues,” in Proceedings of the 2015 Workshop on Mobile Big Data (Mobidata ’15), June 2015, pp. 37–42.
Bibliography

[58] M. Mukherjee, L. Shu, and D. Wang, “Survey of fog computing: Fundamental, network applications, and research challenges,” *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 1826–1857, March 2018.

[59] R. Mahmud, K. Ramamohanarao, and R. Buyya, “Fog computing: A taxonomy, survey and future directions,” *Internet of Everything. Internet of Things (Technology, Communications and Computing)*, pp. 103–130, 2018.

[60] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, “All one needs to know about fog computing and related edge computing paradigms: A complete survey,” *Journal of Systems Architecture*, vol. 98, pp. 289–330, September 2019.

[61] Z. Rejiba, X. Masip-Bruin, and E. Marín-Tordera, “A survey on mobility-induced service migration in the fog, edge, and related computing paradigms,” *ACM Computing Surveys*, vol. 52, no. 5, pp. 90:1–90:33, September 2019.

[62] L. Chunlin and L. LaYuan, “Cost and energy aware service provisioning for mobile client in cloud computing environment,” *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1196–1223, April 2015.

[63] X. Luo, “From augmented reality to augmented computing: A look at cloud-mobile convergence,” in *Proceedings of the 2009 International Symposium on Ubiquitous Virtual Reality*, July 2009, pp. 29–32.

[64] N. Z. Naqvi, K. Moens, A. Ramakrishnan, D. Preuveneers, D. Hughes, and Y. Berbers, “To cloud or not to cloud: A context-aware deployment perspective of augmented reality mobile applications,” in *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC ’15)*, April 2015, pp. 555–562.

[65] R. Shea, A. Sun, S. Fu, and J. Liu, “Towards fully offloaded cloud-based ar: Design, implementation and experience,” in *Proceedings of the 8th ACM on Multimedia Systems Conference (MMSys’17)*, June 2017, pp. 321–330.

[66] Z. Huang, W. Li, P. Hui, and C. Peylo, “CloudRidAR: A cloud-based architecture for mobile augmented reality,” in *Proceedings of the 2014 Workshop on Mobile Augmented Reality and Robotic Technology-based Systems (MARS ’14)*, June 2014, pp. 29–34.

[67] E. Ahvar, A.-C. Orgerie, and A. Lébre, “Estimating energy consumption of cloud, fog and edge computing infrastructures,” *IEEE Transactions on Sustainable Computing*, March 2019, Early Access.
[68] M. Trinelli, M. Gallo, M. Rifai, and F. Pianese, “Transparent AR processing acceleration at the edge,” in Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking (EdgeSys ’19), March 2019, pp. 30–35.

[69] M. Aazam and E. N. Huh, “Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT,” in Proceedings of the 2015 IEEE 29th International Conference on Advanced Information Networking and Applications, March 2015, pp. 687–694.

[70] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, “Dynamic service placement for mobile micro-clouds with predicted future costs,” IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 4, pp. 1002–1016, April 2017.

[71] P. Borylo, A. Lason, J. Rzasa, A. Szymanski, and A. Jajszczyk, “Energy-aware fog and cloud interplay supported by wide area software defined networking,” in Proceedings of the 2016 IEEE International Conference on Communications (ICC), May 2016, pp. 1–7.

[72] C. Fricker, F. Guillemin, P. Robert, and G. Thompson, “Analysis of an offloading scheme for data centers in the framework of fog computing,” ACM Transactions on Modeling and Performance Evaluation of Computing Systems, vol. 1, no. 4, pp. 16:1–16:18, September 2016.

[73] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, “Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control,” IEEE Transactions on Computers, vol. 66, no. 5, pp. 810–819, May 2017.

[74] A. R. Zamani, M. Zou, J. Diaz-Montes, I. Petri, O. Rana, A. Anjum, and M. Parashar, “Deadline constrained video analysis via in-transit computational environments,” IEEE Transactions on Services Computing, pp. 59–72, January 2017.

[75] W. Tärneberg, A. Mehta, E. Wadbro, J. Tordsson, J. Eker, M. Kihl, and E. Elmqroth, “Dynamic application placement in the mobile cloud network,” Future Generation Computer Systems, vol. 70, pp. 163–177, May 2017.

[76] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, “LAVEA: Latency-aware video analytics on edge computing platform,” in Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC ’17), October 2017, pp. 15:1–15:13.

[77] Y. Liu, M. J. Lee, and Y. Zheng, “Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system,” IEEE Transactions on Mobile Computing, vol. 15, no. 10, pp. 2398–2410, October 2016.
[78] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, “Mobility-aware application scheduling in fog computing,” IEEE Cloud Computing, vol. 4, no. 2, pp. 26–35, March 2017.

[79] L. Tang, S. He, and Q. Li, “Double-sided bidding mechanism for resource sharing in mobile cloud,” IEEE Transactions on Vehicular Technology, vol. 66, no. 2, pp. 1798–1809, February 2017.

[80] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, “Cost efficient resource management in fog computing supported medical cyber-physical system,” IEEE Transactions on Emerging Topics in Computing, vol. 5, no. 1, pp. 108–119, January 2017.

[81] L. Chen and J. Xu, “Socially trusted collaborative edge computing in ultra dense networks,” in Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC ’17), October 2017, pp. 9:1–9:11.

[82] S. Sardellitti, G. Scutari, and S. Barbarossa, “Joint optimization of radio and computational resources for multicell mobile-edge computing,” IEEE Transactions on Signal and Information Processing over Networks, vol. 1, no. 2, pp. 89–103, June 2015.

[83] A. Elgazar, M. Aazam, and K. Harras, “Edgestore: Leveraging edge devices for mobile storage offloading,” in Proceedings of the 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), December 2018, pp. 56–61.

[84] Q. Zhang, X. Zhang, Q. Zhang, W. Shi, and H. Zhong, “Firework: Big data sharing and processing in collaborative edge environment,” in Proceedings of the 2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), October 2016, pp. 20–25.

[85] F. Ait Salaht, F. Desprez, A. Lebre, C. Prud’homme, and M. Abderraham, “Service placement in fog computing using constraint programming,” in Proceedings of the 2019 IEEE International Conference on Services Computing (SCC), July 2019, pp. 19–27.

[86] E. Borgia, R. Bruno, M. Conti, D. Mascitti, and A. Passarella, “Mobile edge clouds for information-centric IoT services,” in Proceedings of the 2016 IEEE Symposium on Computers and Communication (ISCC), June 2016, pp. 422–428.

[87] J. G. Boubin, N. T. R. Babu, C. Stewart, J. Chumley, and S. Zhang, “Managing edge resources for fully autonomous aerial systems,” in Proceedings of the 4th ACM/IEEE Symposium on Edge Computing (SEC ’19), November 2019, pp. 74–87.
[88] V. De Maio and I. Brandic, “Multi-objective mobile edge provisioning in small cell clouds,” in Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering (ICPE ’19), April 2019, pp. 127–138.

[89] A. J. Fahs and G. Pierre, “Proximity-aware traffic routing in distributed fog computing platforms,” in Proceedings of the 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), May 2019, pp. 478–487.

[90] R. Mahmud, K. Ramamohanarao, and R. Buyya, “Latency-aware application module management for fog computing environments,” ACM Trans. Internet Technol., vol. 19, no. 1, pp. 9:1–9:21, November 2018.

[91] C. Sonmez, A. Ozgovde, and C. Ersoy, “Fuzzy workload orchestration for edge computing,” IEEE Transactions on Network and Service Management, vol. 16, no. 2, pp. 769–782, June 2019.

[92] L. Wang, L. Jiao, J. Li, J. Gedeon, and M. Mühlhäuser, “MOERA: Mobility-agnostic online resource allocation for edge computing,” IEEE Transactions on Mobile Computing, vol. 18, no. 8, pp. 1843–1856, August 2019.

[93] S. Wang, M. Zafer, and K. K. Leung, “Online placement of multi-component applications in edge computing environments,” IEEE Access, vol. 5, pp. 2514–2533, February 2017.

[94] Y. Xia, X. Etchevers, L. Letondeur, A. Lebre, T. Coupaye, and F. Desprez, “Combining heuristics to optimize and scale the placement of IoT applications in the fog,” in Proceedings of the 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), December 2018, pp. 153–163.

[95] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, “Optimized IoT service placement in the fog,” Service Oriented Computing and Applications, vol. 11, no. 4, pp. 427–443, December 2017.

[96] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, “Quality of experience (QoE)-aware placement of applications in fog computing environments,” Journal of Parallel and Distributed Computing, pp. 190–203, October 2019.

[97] T. Ouyang, Z. Zhou, and X. Chen, “Follow Me at the edge: Mobility-aware dynamic service placement for mobile edge computing,” IEEE Journal on Selected Areas in Communications, vol. 36, no. 10, pp. 2333–2345, September 2018.
[98] O. Skarlat, V. Karagiannis, T. Rausch, K. Bachmann, and S. Schulte, “A framework for optimization, service placement, and runtime operation in the fog,” in Proceedings of the IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), December 2018, pp. 164–173.

[99] V. Souza, X. Masip-Bruin, E. Marín-Tordera, S. Sánchez-López, J. García, G. Ren, A. Jukan, and A. J. Ferrer, “Towards a proper service placement in combined fog-to-cloud (F2C) architectures,” Future Generation Computer Systems, vol. 87, pp. 1 – 15, October 2018.

[100] D. Santoro, D. Zozin, D. Pizzolli, F. D. Pellegrini, and S. Cretti, “Foggy: A platform for workload orchestration in a fog computing environment,” in Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom), December 2017, pp. 231–234.

[101] M. Guiguis, Q. Gu, T. Penner, L. Tammineni, T. Langford, A. Rivera-Longoria, A. Johnson, and B. V. Slyke, “Assignment and collaborative execution of tasks on transient clouds,” Annals of Telecommunications, vol. 73, no. 3, pp. 251–261, April 2018.

[102] D. Mascitti, M. Conti, A. Passarella, L. Ricci, and S. K. Das, “Service provisioning in mobile environments through opportunistic computing,” IEEE Transactions on Mobile Computing, vol. 17, no. 12, pp. 2898–2911, December 2018.

[103] Y. Li and S. Wang, “An energy-aware edge server placement algorithm in mobile edge computing,” in Proceedings of the IEEE International Conference on Edge Computing (EDGE), July 2018, pp. 66–73.

[104] J. Plachy, Z. Becvar, P. Mach, R. Marik, and M. Vondra, “Joint positioning of flying base stations and association of users: Evolutionary-based approach,” IEEE Access, vol. 7, pp. 11454–11463, January 2019.

[105] Q. Fan, N. Ansari, and X. Sun, “Energy driven avatar migration in green cloudlet networks,” IEEE Communications Letters, vol. 21, no. 7, pp. 1601–1604, July 2017.

[106] C. L. T. Nguyen, “Autonomous resource management for mobile edge clouds,” Licentiate Thesis, Umeå university, Sweden, September 2019. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-162924

[107] A. Mitibaa, A. Fahim, K. A. Harras, and M. H. Ammar, “Towards resource sharing in mobile device clouds: Power balancing across mobile devices,” in Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing (MCC ’13), August 2013, pp. 51–56.
[108] A. P. Bianzino, M. Asplund, E. J. Vergara, and S. Nadjm-Tehrani, “Co-operative proxies: Optimally trading energy and quality of service in mobile devices,” *Computer Networks*, vol. 75, pp. 297–312, December 2014.

[109] J. Plachy, Z. Becvar, and E. C. Strinati, “Dynamic resource allocation exploiting mobility prediction in mobile edge computing,” in *Proceedings of the 2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, September 2016, pp. 1–6.

[110] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, “The role of cloudlets in hostile environments,” *IEEE Pervasive Computing*, vol. 12, no. 4, pp. 40–49, October 2013.

[111] K. Toczé, N. Schmitt, I. Brandic, A. Aral, and S. Nadjm-Tehrani, “Towards edge benchmarking: A methodology for characterizing edge workloads,” in *Proceedings of the 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, June 2019, pp. 70–71.

[112] K. Toczé and S. Nadjm-Tehrani, “A taxonomy for management and optimization of multiple resources in edge computing,” *Wireless Communications and Mobile Computing*, vol. 2018, pp. 7 476 201:1–7 476 201:23, June 2018.

[113] B. Confais, A. Lebre, and B. Parrein, “An object store service for a fog/edge computing infrastructure based on IPFS and a scale-out NAS,” in *Proceedings of the 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, May 2017, pp. 41–50.

[114] J. Wang, J. Pan, and F. Esposito, “Elastic urban video surveillance system using edge computing,” in *Proceedings of the Workshop on Smart Internet of Things (SmartIoT ’17)*, October 2017, pp. 7:1–7:6.

[115] H. R. Arkian, A. Diyanat, and A. Pourkhalili, “MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications,” *Journal of Network and Computer Applications*, vol. 82, pp. 152 – 165, March 2017.

[116] C. Sonmez, A. Ozgovde, and C. Ersoy, “Edgecloudsim: An environment for performance evaluation of edge computing systems,” *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, pp. e3493:1–e3493:17, August 2018.

[117] J. McChesney, N. Wang, A. Tanwer, E. de Lara, and B. Varghese, “DeFog: Fog computing benchmarks,” in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing (SEC ’19)*, November 2019, p. 47–58.
[118] K. Intharawijitr, K. Iida, and H. Koga, “Analysis of fog model considering computing and communication latency in 5G cellular networks,” in Proceedings of the 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), March 2016, pp. 1–4.

[119] G. Lee, W. Saad, and M. Bennis, “An online secretary framework for fog network formation with minimal latency,” in Proceedings of the 2017 IEEE International Conference on Communications (ICC), May 2017, pp. 1–6.

[120] S. Shen, V. v. Beek, and A. Iosup, “Statistical characterization of business-critical workloads hosted in cloud datacenters,” in Proceedings of the 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 2015, pp. 465–474.

[121] S. Talluri, A. Luszcak, C. L. Abad, and A. Iosup, “Characterization of a big data storage workload in the cloud,” in Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering (ICPE ’19), April 2019, pp. 33–44.

[122] S. Ahn, M. Gorlatova, P. Naghizadeh, M. Chiang, and P. Mittal, “Adaptive fog-based output security for augmented reality,” in Proceedings of the 2018 Morning Workshop on Virtual Reality and Augmented Reality Network (VR/AR Network ’18), August 2018, pp. 1–6.

[123] S. Ahn, M. Gorlatova, P. Naghizadeh, and M. Chiang, “Personalized augmented reality via fog-based imitation learning,” in Proceedings of the Workshop on Fog Computing and the IoT (IoT-Fog ’19), April 2019, pp. 11–15.

[124] L. Zhang, A. Sun, R. Shea, J. Liu, and M. Zhang, “Rendering multiparty mobile augmented reality from edge,” in Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV ‘19), June 2019, pp. 67–72.

[125] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, “Towards wearable cognitive assistance,” in Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys ’14), June 2014, pp. 68–81.

[126] W. Zhang, B. Han, and P. Hui, “Jaguar: Low latency mobile augmented reality with flexible tracking,” in Proceedings of the 26th ACM International Conference on Multimedia (MM ’18), October 2018, pp. 355–363.

[127] X. Qiao, P. Ren, S. Dustdar, and J. Chen, “A new era for web AR with mobile edge computing,” IEEE Internet Computing, vol. 22, no. 4, pp. 46–55, August 2018.
BIBLIOGRAPHY

[128] V. Fernández, J. M. Orduña, and P. Morillo, “Server implementations for improving the performance of car systems based on mobile phones,” *Journal of Network and Computer Applications*, vol. 44, pp. 72–82, September 2014.

[129] W.-J. Hsu, K. Merchant, H.-W. Shu, C.-H. Hsu, and A. Helmy, “Weighted waypoint mobility model and its impact on ad hoc networks,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 1, pp. 59–63, January 2005.

[130] Z. Xiao, W. Song, and Q. Chen, “Dynamic resource allocation using virtual machines for cloud computing environment,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1107–1117, June 2013.

[131] C. Sonmez, A. Ozgovde, and C. Ersoy, “EdgeCloudSim: An environment for performance evaluation of edge computing systems,” in *Proceedings of the Second International Conference on Fog and Mobile Edge Computing (FMEC)*, May 2017, pp. 39–44.

[132] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, January 2011.

[133] B. Büchel and F. Corman, “Modelling probability distributions of public transport travel time components,” in *18th Swiss Transport Research Conference (STRC)*, May 2018.

[134] S. Shankland. (2018, December) How 5G aims to end network latency. [Online]. Available: https://www.cnet.com/news/how-5g-aims-to-end-network-latency-response-time/

[135] J. Lindqvist, “Edge computing for mixed reality,” Master’s thesis, Linköping University, October 2019. [Online]. Available: http://urn.kb.se/resolve?urn:nbn:se:liu:diva-162133

[136] K. Tateno, F. Tombari, I. Laina, and N. Navab, “CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction,” in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 6565–6574.

[137] J. Engel, T. Schöps, and D. Cremers, “LSD-SLAM: Large-scale direct monocular SLAM,” in *Computer Vision – ECCV 2014*, September 2014, pp. 834–849.
[138] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras,” IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255–1262, October 2017.

[139] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Luján, S. Furber, A. J. Davison, P. H. J. Kelly, and M. F. P. O’Boyle, “SLAMBench2: Multi-objective head-to-head benchmarking for visual SLAM,” in Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), May 2018, pp. 1–8.

[140] PassMark Software. (2019) Passmark CPU benchmarks - high mid range CPUs. Accessed 19th July 2019. [Online]. Available: https://www.cpubenchmark.net/mid_range_cpus.html

[141] PassMark Software. (2019) Passmark Intel vs AMD CPU benchmarks - high end. Accessed 19th July 2019. [Online]. Available: https://www.cpubenchmark.net/high_end_cpus.html

[142] R. B. Miller, “Response time in man-computer conversational transactions,” in Proceedings of the Fall Joint Computer Conference (AFIPS ’68 (Fall, part I)), December 1968, pp. 267–277.

[143] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, “Cloud gaming: architecture and performance,” IEEE Network, vol. 27, no. 4, pp. 16–21, July 2013.

[144] A. Das, S. Patterson, and M. Wittie, “Edgebench: Benchmarking edge computing platforms,” in Proceedings of the 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), December 2018, pp. 175–180.

[145] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, “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, vol. 47, no. 9, pp. 1275–1296, June 2017.

[146] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, “Versatile, scalable, and accurate simulation of distributed applications and platforms,” Journal of Parallel and Distributed Computing, vol. 74, no. 10, pp. 2899–2917, October 2014.

[147] A. Brogi, S. Forti, and A. Ibrahim, “How to best deploy your fog applications, probably,” in Proceedings of the 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), May 2017, pp. 105–114.

[148] K. Toczé and S. Nadjm-Tehrani, “ORCH: Distributed orchestration framework using mobile edge devices,” in Proceedings of the 2019 IEEE
3rd International Conference on Fog and Edge Computing (ICFEC), May 2019, pp. 1–10.

[149] D. J. Hand, Statistics. a very short introduction, ser. Very short introduc-
tions. Oxford University Press, October 2008.
Department of Computer and Information Science  
Linköpings universitet  
Licentiate Theses  
Linköpings Studies in Science and Technology  
Faculty of Arts and Sciences

No 17Vojin Plavsic: Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)

No 28Arne Jönsson, Mikael Patel: An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.

No 29Johnny Eckerland: Retargeting of an Incremental Code Generator, 1984.

No 48Henrik Nordin: On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.

No 52Zeho Peng: Steps Towards the Formalization of Designing VLSI Systems, 1985.

No 60Johan Fagerström: Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.

No 71Jalal Maleki: ICONstraint, A Dependency Directed Constraint Maintenance System, 1987.

No 72Tony Larsson: On the Specification and Verification of VLSI Systems, 1986.

No 73Ola Strömberg: A Structure Editor for Documents and Programs, 1986.

No 74Christos Levcopoulos: New Results about the Approximation Behavior of the Greedy Triangulation, 1986.

No 104Shamsul I. Chowdhury: Statistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.

No 108Rober Bilios: Incremental Scanning and Token-Based Editing, 1987.

No 111Hans Block: SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.

No 113Ralph Rönquist: Network and Lattice Based Approaches to the Representation of Knowledge, 1987.

No 116Maria Kamkar, Nahid Shahmehri: On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.

No 150Tim Hansen: Defining and Controlling Actions in a Mechanical System, 1988.

No 165Jonas Löwgren: Supporting Design and Management of Expert System User Interfaces, 1989.

No 166Ola Petersson: On Adaptive Sorting in Sequential and Parallel Models, 1989.

No 174Yngve Larsson: Dynamic Configuration in a Distributed Environment, 1989.

No 177Peter Aberg: Design of a Multiple View Presentation and Interaction Manager, 1989.

No 181Henrik Eriksson: A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.

No 184Ivan Rankin: The Deep Generation of Text in Expert Critiquing Systems, 1989.

No 187Samin Nasim-Tehrani: Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.

No 189Magnus Merkel: Temporal Information in Natural Language, 1989.

No 196Ulf Nilsson: A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.

No 197Staffan Bonnier: Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.

No 203Christer Hansson: A Prototype System for Logical Reasoning about Time and Action, 1990.

No 212Björn Fjellborg: An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.

No 237Tomas Sokolnicki: Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.

No 250Lars Strömberg: Postmortem Debugging of Distributed Systems, 1990.

No 259Torbjörn Näslund: SLDEA-Resolution - Computing Answers for Negative Queries, 1990.

No 260Peter D. Holmes: Using Connectivity Graphs to Support Map-Related Reasoning, 1991.

No 283Olof Johansson: Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge Bases, 1991.

No 298Rolf G Larsson: Aktivitetsbaserad kalkylering i ett nytt ekonomsystem, 1991.

No 318Lena Strömberg: Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.

No 319Mikael Pettersson: DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.

No 326Andreas Kågedal: Logic Programming with External Procedures: an Implementation, 1992.

No 328Patrick Lambris: Aspects of Version Management of Composite Objects, 1992.

No 333Xinli Gu: Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.

No 335Torbjörn Näslund: On the Role of Evaluations in Iterative Development of Managerial Support Systems, 1992.

No 348Ulf Cedering: Industrial Software Development - a Case Study, 1992.

No 352Magnus Morin: Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.

No 371Mehran Noghahab: Evaluation of Strategic Investments in Information Technology, 1993.

No 378Mats Larsson: A Transformational Approach to Formal Digital System Design, 1993.
Johan Ringström: Compiler Generation for Parallel Languages from Denotational Specifications, 1993.
Michael Jansson: Propagation of Change in an Intelligent Information System, 1993.
Jonni Harris: An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.
Per Österling: Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.
Johan Boye: Dependency-based Groundness Analysis of Functional Logic Programs, 1993.
Lars Degersted: Tabulated Resolution for Well Founded Semantics, 1993.
Anna Moberg: Separation of foretagsledning and finansiering - fallstudier av foretagsledarurk ur ett agent-teoretiskt perspektiv, 1994.
Camilla Sjöström: Revision och lagrelering - ett historiskt perspektiv, 1994.
Cecilia Sjöberg: Voices in Design: Argumentation in Participatory Development, 1994.
Lars Viklund: Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
Peter Loborg: Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94: Owen Eriksson: Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och sannskapande perspektiv, 1994.
FHS 4/94: Karin Pettersson: Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
Lars Poignant: Informationsteknologi och foretagssetablering - Effekter på produktivitet och region, 1994.
Gustav Fahl: Object Views of Relational Data in Multidatabase Systems, 1994.
Henrik Nilsson: A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
Jonas Lind: Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
Martin Sköld: Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
Pär Carlshamre: A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94: Stefan Cronholm: Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
Mikael Lindvall: A Study of Traceability in Object-Oriented Systems Development, 1994.
Fredrik Nilsson: Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
Hans Ösén: Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
Lars Karbsson: Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
Ulf Söderman: On Conceptual Modelling of Mode Switching Systems, 1995.
Choong Hoo Yi: Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
Bo Lagerström: Successive resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.
Peter Jonsson: Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95: Anders Avdic: Arbetsintegrerad systemutveckling med kalkylprogram, 1995.
Eva I. Ragnemalm: Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
Eva Toller: Contributions to Parallel Multidaparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
Erik Stoy: A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
Johan Herber: Environment Support for Building Structured Mathematical Models, 1995.
Stefan Svenberg: Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
Hec-Cheol Kim: Prediction and Postdiction under Uncertainty, 1995.
Dan Frisödt: Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
Malin Bergvall: Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
Joachim Karlschon: Towards a Strategy for Software Requirements Selection, 1995.
Jakob Axelsson: Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
Göran Forslund: Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
Jörgen Andersson: Bilder av småföretagares ekonomistyrning, 1995.
Staffan Flodin: Efficient Management of Object-Oriented Queries with Late Binding, 1995.
Vadim Engelson: An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
Magnaus Werner: Multidatabase Integration using Polymorphic Queries and Views, 1996.
Mikael Lind: Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
Jonas Hallberg: High-Level Synthesis under Local Timing Constraints, 1996.
Kristina Larsen: Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag, 1996.
Mikael Johansson: Quality Functions for Requirements Engineering Methods, 1996.
Patrik Nordling: The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
Anders Ekman: Exploration of Polygonal Environments, 1996.
Nidas Andersson: Compilation of Mathematical Models to Parallel Code, 1996.
Verksamhetsförändringar i samband med IS-utveckling av affärsrelationer och informationssystem, 2000.

Susanne Odar: Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationsåtkomst och skadegörelse av informationssystem, 2000.

Lin Han: Optionsprogram för anställda - en studie av svenska börsföretag, 2001.

Bourhane Kadmiry: Automatisera informationskrever i organisation, 2000.

Fredrik Karlsson: Towards Rational Unification of Declarative Equation-Based Languages, 2002.

Pär J. Ågerfalk: Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.

Charlotte Bjarkegren: Learning for the next project - Bärers and barriers in knowledge transfer within an organisation, 1999.

Håkan Nilsson: Informationsteknik som drivkraft i granskningsprocessen - En studie af fyra revisionsbyråer, 2000.

Erik Berglund: Use-Oriented Documentation in Software Development, 1999.

Klas Gäre: Verksamhetsförändringar i samband med IS-införande, 1999.

Anders Subotie: Software Quality Inspection, 1999.

Svein Bergum: Managerial communication in telework, 2000.

Flavius Gruian: Energy-Aware Design of Digital Systems, 2000.

Karin Hedström: Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.

Linda Askénäs: Affärsstystem - En studie om teknikens aktiva och passiva roll i en organisation, 2000.

Jean Paul Meynard: Control of industrial robots through high-level task programming, 2000.

Lars Hult: Publika Gränsytor - ett designexempel, 2000.

Paul Pop: Scheduling and Communication Synthesis for Distributed Real-Time Systems, 2000.

Goran Hultgren: Nätverksskrapid förändringsanalys - perspektiv och metoder som stöd för förståelse och utveckling av affärsrelationer och informationsstystem, 2000.

Mikael Cäker: TDV stöd för strategiska beslut, en studie av datorimplementerade modeller av verksamhet som stöd för beslut om anskaffning av JAS 1982, 2002.

Ewa Braf: Organisationers kunskapsverksamhet - en kritisk studie av "knowledge management", 2000.

Henrik Lindberg: Webbaserade affärsprocesser - Mått och begränsningar, 2000.

Benneth Christiansson: Att komponentbaserad informationssystem - Vad säger teori och praktik?, 2000.

Ola Pettersson: Deliberation in a Mobile Robot, 2000.

Dan Lawesson: Towards Behavioral Model Fault Isolation for Object Oriented Control Systems, 2000.

Johan Moe: Execution Tracing of Large Distributed Systems, 2001.

Yuxiao Zhao: XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement, 2001.

Annika Flycht-Eriksson: Domain Knowledge Representation in Information-providing Dialogue systems, 2001.

Per-Arne Segerkvist: Webbaserade imaginära organisationers samverkansformer: Informationssystemmaktritur och aktörssamverkan som förutsättningar för affärsprocesser, 2001.

Stefan Svarén: Styrning av investeringar i divisionaliserade företag - ett konkurrencespektiv, 2001.

Lin Han: Secure and Scalable E-Service Software Delivery, 2001.

Emma Hansson: Optionsprogram för anställda - en studie av svenska börsföretag, 2001.

Susanne Ödström: IT som stöd för strategiska beslut, en studie av datorimplementerade modeller av verksamhet som stöd för beslut om anskaffning av JAS 1982, 2002.

Stefan Holgersson: IT-system och filtrering av verksamhetskunskap - kvalitetsproblem vid analyser och beslutstagna som bygger på uppgifter hämtade från polisens IT-system, 2001.

Per Oscarsson: Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationsåtkomst och dess hantering, 2001.

Luis Alejandro Cortes: A Petri Net Based Modeling and Verification Technique for Real-Time Embedded Systems, 2001.

Niklas Sandell: Redovisning i skuggan av en bankkris - Vårdning av fastigheter, 2001.

Fredrik Elg: Ett dynamiskt perspektiv på individuella skillnader av heuristisk kompetens, intelligens, mentala modeller, mål och konfidenståntal av mikrovärlden Mero, 2002.

Peter Aromsson: Automatic Parallelization of Simulation Code from Equation Based Simulation Languages, 2002.

Bourhane Kadmiry: Fuzzy Control of Unmanned Helicopter, 2002.

Patrik Haslum: Declarative Analysis of Dialogue systems, 2001.

Robert Severson: On the instruments of governance - a law & economics study of capital instruments in limited liability companies, 2002.

Johan Petersson: Lokala elektroniska marknadssplatser - informationssystem för platsbundna affärer, 2002.

Peter Bonus: Debugging and Structural Analysis of Declarative Equation-Based Languages, 2002.

Gert Jervan: High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems, 2002.

Fredrika Berglund: Management Control and Strategy - a Case Study of Pharmaceutical Drug Development, 2002.

Fredrik Karlsson: Meta-Method for Method Configuration - A Rational Unified Process Case, 2002.

Sorin Manolache: Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times, 2002.

Diana Szentivián: Performance and Availability Trade-offs in Fault-Tolerant Middleware, 2002.

Iakov Nakhimovski: Modeling and Simulation of Contacting Flexible Bodies in Multibody Systems, 2002.

Levon Saldamli: PDEModelica - Towards a High-Level Language for Modeling with Partial Differential Equations, 2002.

Almut Herzog: Secure Execution Environment for Java Electronic Services, 2002.
Andreas Hansson: Increasing the Storage Capacity of Recursive Auto-associative Memory by Segmenting Data, 2005.

Nicklas Bergfeldt: Towards Detached Communication for Robot Cooperation, 2005.

Dennis Maciuszek: Towards Dependable Virtual Companions for Later Life, 2005.

Beatrice Aalenljung: Decision-making in the Requirements Engineering Process: A Human-centered Approach, 2005.

Anders Larsson: System-on-Chip Test Scheduling and Test Infrastructure Design, 2005.

John Wilander: Policy and Implementation Assurance for Software Security, 2005.

Andreas Käll: Oversättningar av en managementmodell - En studie av införandet av Balanced Scorecard i ett landsting, 2005.

He Tax: Aligning and Merging Biomedical Ontologies, 2006.

Artur Wilk: Descriptive Types for XML Query Language Xcerpt, 2006.

Per Olof Pettersson: Sampling-based Path Planning for an Autonomous Helicopter, 2006.

Kalle Burbeck: Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks, 2006.

Daniela Mihalcescu: Implementation Methodology in Action: A Study of an Enterprise Systems Implementation Methodology, 2006.

Jörgen Skågeby: Public and Non-public gifting on the Internet, 2006.

Karolina Eliasson: The Use of Case-Based Reasoning in a Human-Robot Dialog System, 2006.

Misook Park: Managing Competence Development Programs in a Cross-Cultural Organisation - What are the Barriers and Enablers, 2006.

FiF-a 90: a 90

Amra Hallilovic: Ett praktikperspektiv på hantering av mjukvarukomponenter, 2006.

FiF-a 91: a 91

Hanna Broberg: Verksamhetsanpassade IT-stöd - Designteori och metod, 2006.

Robert Kaminski: Towards an XML Document Restructuring Framework, 2006.

Jiri Trnka: Prerequisites for data sharing in emergency management, 2007.

Björn Hägglund: A Framework for Designing Constraint Stores, 2007.

Daniel Andreasson: Slack-Time Aware Dynamic Routing Schemes for On-Chip Networks, 2007.

Magnus Ingmarsson: Modelling User Tasks and Intentions for Service Discovery in Ubiquitous Computing, 2007.

Gustaf Svedjemo: Ontology as Conceptual Schema when Modelling Historical Maps for Database Storage, 2007.

Gianpaolo Conte: Navigation Functionality for an Autonomous UAV Helicopter, 2007.

Ola Leifler: User-Centric Critiquing in Command and Control: The DKExpert and ComPlan Approaches, 2007.

Henrik Svensson: Embodied simulation as off-line representation, 2007.

Zhiyuan He: System-on-Chip Test Scheduling with Defect-Probability and Temperature Considerations, 2007.

Jonas Elmqvist: Components, Safety Interfaces and Compositional Analysis, 2007.

Håkan Sundblad: Question Classification in Question Answering Systems, 2007.

Magnus Lundqvist: Information Demand and Use: Improving Information Flow within Small-scale Business Contexts, 2007.

Martin Magnusson: Deductive Planning and Composite Actions in Temporal Action Logic, 2007.

Mikael Asplund: Restoring Consistency after Network Partitions, 2007.

Martin Fransson: Towards Individualised Drug Dosage - General Methods and Case Studies, 2007.

Karin Camara: A Visual Query Language Served by a Multi-sensor Environment, 2007.

David Broman: Safety, Security, and Semantic Aspects of Equation-Based Object-Oriented Languages and Environments, 2007.

Mikhail Chalabine: Invasive Interactive Parallelization, 2007.

Susanna Nilsson: A Holistic Approach to Usability Evaluations of Mixed Reality Systems, 2008.

Shanai Ardi: A Model and Implementation of a Security Plug-in for the Software Life Cycle, 2008.

Erik Kuiper: Mobility and Routing in a Delay-tolerant Network of Unmanned Aerial Vehicles, 2008.

Jana Rambusch: Situated Play, 2008.

Martin Karresand: Completing the Picture - Fragments and Back Again, 2008.

Per Nyblom: Dynamic Abstraction for Interleaved Task Planning and Execution, 2008.

Fredrik Lantz: Terrain Object Recognition and Context Fusion for Decision Support, 2008.

Martin Östlund: Assistance Plus: 3D-mediated Advice-giving on Pharmaceutical Products, 2008.

Håkan Lundvall: Automatic Parallelization using Pipelining for Equation-Based Simulation Languages, 2008.

Mirko Thorstensson: Using Observers for Model Based Data Collection in Distributed Tactical Operations, 2008.

Bahloh Rahimi: Implementation of Health Information Systems, 2008.

Maria Holmqvist: Word Alignment by Re-using Parallel Phrases, 2008.

Mattias Eriksson: Integrated Software Pipelining, 2009.

Annika Öhgren: Towards an Ontology Development Methodology for Small and Medium-sized Enterprises, 2009.

Rickard Holmmark: Deadlock Free Routing in Mesh Networks on Chip with Regions, 2009.

Sara Stymne: Compound Processing for Phrase-Based Statistical Machine Translation, 2009.

Tommy Elqvist: Supporting Scientific Collaboration through Workflows and Provenance, 2009.

Fabian Segelström: Visualisations in Service Design, 2010.

Min Bao: System Level Techniques for Temperature-Aware Energy Optimization, 2010.

Mohammad Saifullah: Exploring Biologically Inspired Interactive Networks for Object Recognition, 2011.
Qiang Liu: Dealing with Missing Mappings and Structure in a Network of Ontologies, 2011.
Ruxandra Pop: Mapping Concurrent Applications to Multiprocessor Systems with Multithreaded Processors and Network on Chip-Based Interconnections, 2011.
Per-Magnus Olsson: Positioning Algorithms for Surveillance Using Unmanned Aerial Vehicles, 2011.
Anna Vapen: Contributions to Web Authentication for Untrusted Computers, 2011.
Loove Broms: Sustainable Interactions: Studies in the Design of Energy Awareness Artifacts, 2011.
Joh Söding: A Conceptualising Prototypes in Service Design, 2011.
Håkan Warnquist: Computer-Assisted Troubleshooting for Efficient Off-board Diagnosis, 2011.
Jakob Rosén: Predictable Real-Time Applications on Multiprocessor Systems-on-Chip, 2011.
Usman Dastgir: Skeleton Programming for Heterogeneous GPU-based Systems, 2011.
David Lands: Complex Task Allocation for Delegation: From Theory to Practice, 2011.
Krisian Staväker: Contributions to Parallel Simulation of Equation-Based Models on Graphics Processing Units, 2011.
Mariusz Wzorek: Selected Aspects of Navigation and Path Planning in Unmanned Aircraft Systems, 2011.
Piotr Kuol: Increasing Autonomy of Unmanned Aircraft Systems Through the Use of Imaging Sensors, 2011.
Anders Carstensen: The Evolution of the Connector View Concept: Enterprise Models for Interoperability Solutions in the Extended Enterprise, 2011.
Jody Foo: Computational Terminology: Exploring Bilingual and Monolingual Term Extraction, 2012.
Anders Fröberg: Models and Tools for Distributed User Interface Development, 2012.
Dimitar Nikolov: Optimizing Fault Tolerance for Real-Time Systems, 2012.
Dennis Andersson: Mission Experience: How to Model and Capture it to Enable Vicarious Learning, 2013.
Massimiliano Ractil: Anomaly Detection and its Adaptation: Studies on Cyber-physical Systems, 2013.
Banafsheh Khademhosseinieh: Towards an Approach for Efficiency Evaluation of Enterprise Modeling Methods, 2013.
Amy Rankin: Resilience in High Risk Work: Analysing Adaptive Performance, 2013.
Martín Sjölund: Tools for Understanding, Debugging, and Simulation Performance Improvement of Equation-Based Models, 2013.
Karl Hammar: Towards an Ontology Design Pattern Quality Model, 2013.
Maria Vasilevskaia: Designing Security-enhanced Embedded Systems: Bridging Two Islands of Expertise, 2013.
Ekhivotz Vergara: Exploiting Energy Awareness in Mobile Communication, 2013.
Valentina Ivanova: Integration of Ontology Alignment and Ontology Debugging for Taxonomy Networks, 2014.
Dag Sonntag: A Study of Chain Graph Interpretations, 2014.
Kiril Kiryazov: Grounding Emotion Appraisal in Autonomous Humanoids, 2014.
Zlatan Dragić: Completing the Is-a Structure in Description Logics Ontologies, 2014.
Erik Hansson: Code Generation and Global Optimization Techniques for a Reconfigurable PRAM-NUMA Multicore Architecture, 2014.
Nicolas Melot: Energy-Efficient Computing over Streams with Massively Parallel Architectures, 2015.
Mahder Gebremedhin: Automatic and Explicit Parallelization Approaches for Mathematical Simulation Models, 2015.
Mikael Nilsson: Efficient Temporal Reasoning with Uncertainty, 2015.
Vladislavs Jahundovics: Automatic Verification of Parameterized Sytems by Over-Approximation, 2015.
Camilla Kirkegaard: Adding Challenge to a Teachable Agent in a Virtual Learning Environment, 2016.
Vengatanathan Krishnamoorthy: Efficient and Scalable Content Delivery of Linear and Interactive Branched Videos, 2016.
Andreas Löwenmark: Timing Predictability in Future Multi-Core Avionics Systems, 2017.
Anders Andersson: Extensions for Distributed Moving Base Driving Simulators, 2017.
Olov Andersson: Methods for Scalable and Safe Robot Learning, 2017.
Robin Keskiärikä: Towards Semantically Enabled Complex Event Processing, 2017.
Daniel de Long: Spatio-Temporal Stream Reasoning with Adaptive State Stream Generation, 2017.
Johan Falkenjuk: Towards a Model of General Text Complexity for Swedish, 2018.
Magdalena Granåsen: Exploring C2 Capability and Effectiveness in Challenging Environments: Interorganizational Crisis Management, Military Operations and Cyber Defence, 2019.
Klervie Toczé: Latency-aware Resource Management at the Edge, 2020.
