TECHNISCHE
UNIVERSITÄT
DARMSTADT

Urban Edge Computing

Vom Fachbereich Informatik
der Technischen Universität Darmstadt

Dissertation

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

Eingereicht von:
Julien Alexander Gedeon
geboren am 4. August 1987 in Frankfurt am Main

Tag der Einreichung:	18.06.2020
Tag der Disputation:	30.07.2020
Erstreferent:	Prof. Dr. Max Mühlhäuser
Korreferent:	Prof. Dr. Christian Becker

Darmstadt 2020
Hochschulkennziffer D 17

Julien Alexander Gedeon: *Urban Edge Computing*
Darmstadt, Technische Universität Darmstadt

Jahr der Veröffentlichung der Dissertation auf TUprints: 2020
URN: urn:nbn:de:tuda-tuprints-13362
URI: `https://tuprints.ulb.tu-darmstadt.de/id/eprint/13362`
Tag der mündlichen Prüfung: 30.07.2020

Abstract

The new paradigm of *Edge Computing* aims to bring resources for storage and computations closer to end devices, alleviating stress on core networks and enabling low-latency mobile applications. While Cloud Computing carries out processing in large centralized data centers, Edge Computing leverages smaller-scale resources—often termed *cloudlets*—in the vicinity of users. Edge Computing is expected to support novel applications (e.g., mobile augmented reality) and the growing number of connected devices (e.g., from the domain of the *Internet of Things*). Today, however, we lack essential building blocks for the widespread public availability of Edge Computing, especially in urban environments. This thesis makes several contributions to the understanding, planning, deployment, and operation of *Urban Edge Computing* infrastructures. We start from a broad perspective by conducting a thorough analysis of the field of Edge Computing, systematizing use cases, discussing potential benefits, and analyzing the potential of Edge Computing for different types of applications.

We propose re-using existing physical infrastructures (cellular base stations, WiFi routers, and augmented street lamps) in an urban environment to provide computing resources by upgrading those infrastructures with cloudlets. On the basis of a real-world dataset containing the location of those infrastructures and mobility traces of two mobile applications, we conduct the first large-scale measurement study of *urban cloudlet coverage* with four different metrics for coverage. After having shown the viability of using those existing infrastructures in an urban environment, we make an algorithmic contribution to the problem of which locations to upgrade with cloudlets, given the heterogeneous nature (with regards to communication range, computing resources, and costs) of the underlying infrastructure. Our proposed solution operates locally on grid cells and is able to adapt to the desired tradeoff between the quality of service and costs for the deployment. Using a simulation experiment on the same mobility traces, we show the effectiveness of our strategy.

Existing mechanisms for computation offloading typically achieve loose coupling between the client device and the computing resources by requiring prior transfers of heavyweight execution environments. In light of this deficiency, we propose the concept of *store-based microservice onloading*, embedded in a flexible runtime environment for Edge Computing. Our runtime environment operates on a microservice-level granularity and those services are made available in a repository—the *microservice store*—and, upon request from a client, transferred from the store to execution agents at the edge. Furthermore, our Edge Computing runtime is able to share running instances with multiple users and supports the seamless definition and execution of service chains through distributed message queues. Empirical measurements of the implemented approach showed up to 13 times reduction in the end-to-end latency and energy savings of up to 94 % for the mobile device.

We provide three contributions regarding strategies and adaptations of an Edge Computing system at runtime. Existing strategies for the placement of data and computation components are not adapted to the requirements of a heterogeneous (e.g., with regards to varying resources) edge environment. The placement of functional parts of an application is a core component of runtime decisions. This problem is computationally hard and has been insufficiently explored for service chains whose topologies are typical for Edge Computing environments (e.g., with regards to the location of data sources and sinks). To this end, we present two classes of heuristics that make the problem more tractable. We implement representatives for each class and show how they substantially reduce the time it takes to find a solution to the placement problem, while introducing only a small optimality gap. The placement of data (e.g., such captured by mobile devices) in Edge Computing should take into account the user's context and the possible intent of sharing this data. Especially in the case of overloaded networks, e.g., during large-scale events, edge infrastructure can be beneficial for data storage and local dissemination. To address this challenge, we propose *vStore*, a middleware that—based on a set of rules—decouples applications from pre-defined storage locations in the cloud. We report on results from a field study with a demonstration application, showing that we were able to reduce cloud storage in favor of proximate micro-storage at the edge.

As a final contribution, we explore the adaptation possibilities of microservices themselves. We suggest to make microservices adaptable in three dimensions: (i) in the algorithms they use to perform a certain task, (ii) in their parameters, and (iii) in auxiliary data that is required. These adaptations can be leveraged to trade a faster execution time for a decreased quality of the computation (e.g., by producing more inaccurate or partly wrong results). We argue that this is an important building block to be included in an Edge Computing system in view of both constrained resources and strict requirements on computation latencies. We conceptualize an adaptable microservice execution framework and define the problem of choosing the service variant, building upon the design of our previously introduced Edge Computing runtime environment. For a case study, we implement representative examples (e.g., in the field of computer vision and image processing) and outline the practical influence of the abovementioned tradeoff.

In conclusion, this dissertation systematically analyzes the field of Urban Edge Computing, thereby contributing to its general understanding. Our contributions provide several important building blocks for the realization of a public Edge Computing infrastructure in an urban environment.

Zusammenfassung

Das neue Paradigma des *Edge Computing* zielt darauf ab, Ressourcen für Datenspeicherung und Berechnungen näher an Endgeräte zu verlagern, um so die Belastungen in den Kernnetzen zu verringern und geringe Latenzen für mobile Anwendungen zu ermöglichen. Während bei Cloud Computing die Datenverarbeitung in großen, zentralisierten Rechenzentren erfolgt, nutzt Edge Computing kleinere opportunistische Ressourcen – oftmals als *Cloudlets* bezeichnet – in der Nähe der Benutzer. Es wird davon ausgegangen, dass Edge Computing sowohl neuartige Anwendungen (z.B. Mobile Augmented Reality) ermöglichen, als auch die wachsende Anzahl von vernetzen Geräten (z.B. im Umfeld des *Internet der Dinge*) unterstützen wird. Heute fehlen jedoch wesentliche Bausteine für eine breite, allgemeine Verfügbarkeit von Edge Computing, insbesondere in urbanen Umgebungen. Die vorliegende Dissertation liefert mehrere Beiträge zum Verständnis, zur Planung, zur Bereitstellung, sowie zum Betrieb von *Urban-Edge-Computing*-Infrastrukturen. Wir nehmen zunächst eine breite Perspektive ein, indem wir das Forschungsfeld des Edge Computing eingrenzen, Anwendungsfälle systematisieren, die Vorteile von Edge Computing diskutieren und dessen Potenzial für verschiedene Arten von Anwendungen analysieren.

Wir schlagen vor, bestehende physische Infrastrukturen (Mobilfunk-Basisstationen, WiFi-Router und neuartige Straßenlaternen) in einer städtischen Umgebung zur Bereitstellung von Rechenressourcen zu verwenden, indem diese Infrastrukturen mit Cloudlets aufgerüstet werden. Auf Grundlage eines realen Datensatzes, der die Standorte dieser Infrastrukturen und die Bewegungsdaten zweier mobiler Anwendungen enthält, präsentieren wir erstmals eine groß angelegte Messstudie zur städtischen Cloudlet-Abdeckung. Diese Analyse führen wir auf Grundlage von vier verschiedenen Metriken für Abdeckung durch. Hierdurch zeigen wir die Machbarkeit der Nutzung dieser bestehenden Infrastrukturen in einer städtischen Umgebung für die Bereitstellung von Rechenressourcen durch Cloudlets. Basierend auf diesen Erkenntnissen leisten wir einen algorithmischen Beitrag zur Frage, welche Standorte mit Cloudlets aufgerüstet werden sollten, unter der Annahme, dass die zugrundeliegenden Infrastrukturen heterogen (in Bezug auf Kommunikationsreichweite, Ressourcen und Kosten) sind. Unser Ansatz operiert lokal auf Planquadraten und ist in der Lage, sich an einen variierenden Tradeoff zwischen Dienstgüte und Kosten anzupassen. Mittels eines Simulationsexperiments, das auf den gleichen vorher genannten Bewegungsdaten basiert, zeigen wir die Effektivität unseres Ansatzes.

Bestehende Ansätze für die Auslagerung von Berechnungen sind oft mit einem hohen Aufwand verbunden, weil vor der Ausführung die Übertragung schwergewichtiger Ausführungsumgebungen vom Endgerät erforderlich ist, um eine lose Kopplung zwischen den Endgeräten und den Rechenressourcen zu erreichen. Angesichts dieses Mankos schlagen wir das Konzept des *Microservice Store Onloading* vor und betten dieses in eine flexible Laufzeitumgebung für Edge Computing ein. Diese Laufzeitumgebung nutzt feingranulare Module, sog. *Microservices* für die Ausführung von Berechnungen und hält diese Microservices in einem sog. *Microservice Store* vor. Auf eine Anfrage von Benutzern hin werden die Microservices vom Store direkt auf Agenten am Rande des Netzwerkes übertragen und ausgeführt. Des Weiteren ist die von uns vorgeschlagene Laufzeitumgebung in der Lage, laufende Service-Instanzen zwischen verschiedenen Benutzern zu teilen und ermöglicht über

verteilte Nachrichtenwarteschlangen die Definition und Ausführung von verketteten Microservices. Empirische Messungen des implementierten Ansatzes zeigten eine bis zu 13-mal geringere Ende-zu-Ende-Latenz sowie Energieeinsparungen von bis zu 94 % für die mobilen Client-Geräte.

Wir liefern drei Beiträge zu Entscheidungsstrategien und Anpassungen einer Edge Computing-Ausführungsumgebung zur Laufzeit. Bestehende Strategien zur Platzierung von Daten und Berechnungskomponenten sind nicht an die Anforderungen einer (z.B. in Bezug auf Ressourcen) heterogenen Edge-Computing-Umgebung angepasst. Die Platzierung von funktionalen Teilen einer Anwendung ist eine Kernentscheidung in Ausführungsumgebungen. Dieses Platzierungsproblem ist rechenaufwändig und wurde für Service-Ketten, deren Topologien typisch für Edge-Computing-Umgebungen sind (z.B. im Hinblick auf die Lage von Datenquellen und -Senken im Netzwerk), bisher nur unzureichend untersucht. Aufbauend auf dieser Beobachtung stellen wir zwei Klassen von Heuristiken vor, die das Platzierungsproblem in Edge Computing besser handhabbar machen. Für jede Klasse von Heuristiken implementieren wir Repräsentanten und zeigen, dass unser Ansatz die Zeit, die für die Lösung des Platzierungsproblems benötigt wird, erheblich reduziert. Zudem weisen die so gefundenen Lösungen nur minimale Abweichungen zur optimalen Platzierungsentscheidung auf. Die Platzierung von Daten in Edge Computing (z.B. solcher, die über mobile Endgeräte erfasst werden) sollte idealerweise den aktuellen Kontext des Benutzers sowie das Teilen der Daten berücksichtigen. Insbesondere bei überlasteten Netzwerken, z.B. infolge von Großveranstaltungen, können Edge-Computing-Infrastrukturen nützlich für die Speicherung und Verteilung von Daten sein. Hierzu schlagen wir *vStore* vor, eine Middleware, die – basierend auf einer Menge von Regeln – Anwendungen von ihren vordefinierten Speicherorten in der Cloud entkoppelt. Wir analysieren die Ergebnisse einer Feldstudie, durchgeführt mit einer Beispielanwendung, und zeigen auf, dass unser Ansatz in der Lage ist, Speicherorte von der Cloud an den Rand des Netzes zu verlagern.

Als abschließenden Beitrag untersuchen wir die Anpassungsmöglichkeiten der Microservices selbst. Wir schlagen vor, die Microservices in dreierlei Hinsicht anzupassen: (i) in den Algorithmen, die sie zur Ausführung einer bestimmten Aufgabe verwenden, (ii) in ihren Parametern und (iii) in ggf. für die Ausführung erforderlichen weiteren (Hilfs-)Daten. Diese Anpassungen können z.B. genutzt werden, um eine schnellere Ausführungszeit im Gegenzug für eine verminderte Qualität der Berechnung (z.B. durch ungenauere oder teilweise falsche Ergebnisse) zu erreichen. Wir zeigen auf, dass diese Abwägung ein wichtiger Baustein in Edge-Computing-Umgebungen ist, bedingt sowohl durch die typische Ressourcenknappheit auf der einen, als auch im Hinblick auf strikte Latenzanforderungen auf der anderen Seite. Wir konzipieren eine Laufzeitumgebung, die anpassbare Microservices unterstützt, und definieren das Problem der Auswahl einer konkreten Dienstvariante, wobei wir auf dem Design der zuvor vorgestellten Edge-Computing-Ausführungsumgebung aufbauen. In einer Fallstudie demonstrieren wir anhand von repräsentativen Beispielen (z.B. im Bereich von Computer Vision und Bildverarbeitung) den Praktischen Einfluss der oben genannten Abwägungsentscheidung.

Zusammenfassend bietet diese Dissertation eine systematische Analyse des Forschungsfeldes von Urban Edge Computing. Unsere Beiträge liefern wichtige Bausteine zur Realisierung einer allgemein verfügbaren Edge-Computing-Infrastruktur im urbanen Raum.

Acknowledgments

Completing this thesis would not have been possible without the continuous support and encouragement of my supervisors, colleagues, family, and friends. Thank you for having been by my side on this journey.

First of all, there is of course Max, who gave me the chance to pursue my PhD in the Telecooperation group. Despite your busy schedule at times, I could always count on your support when I needed it most. Thanks for all the inspiring discussions—big and small—and your feedback over the past years. Thanks for creating a working environment where one is able to openly speak their mind. I did not take this for granted. I am also thankful to Christian Becker for his comments and his willingness to act as a co-referee for my thesis.

A big thanks goes to Immanuel Schweizer, who inspired me to do research and, ultimately, recruited me. Working at TK has truly been a pleasure. Michael Stein has been a great colleague in guiding me through the PhD jungle. I'll always value your advice! Big cheers to my other (former) office mates Florian Brandherm, Jens Heuschkel, and Martin Wagner.

Thanks to the following people for having inspired my research and fostered interesting discussions: Carlos Garcia, Alexander Seeliger, Tim Grube, Florian Volk, Tim Neubacher, Sebastian Wagner, Katharina Keller, and Patrick Felka. Sorry to the folks in A316 for my countless interruptions and rants. Speaking of A316, thanks to Jörg Daubert and Rolf Egert for maintaining a healthy supply of refreshments up there. A big thanks has to go to the people who keep TK running, sometimes without being noticed and acknowledged enough: Elke Halla, Elke Reimund, Fabian Herrlich, and Sebastian Alles.

Supervising and working with gifted students has been an amazing part of my time at TK. Following the saying *"If you are the smartest person in the room, you probably are in the wrong room"*, we learned from each other and realized amazing ideas! In this regard, special thanks go out to Disha Bhat, Ali Karpuzoglu, Jeff Krisztinkovics, Nicolas Himmelmann, Karolis Skaisgiris, Alexandra Skogseide, Martin Wagner and Sebastian Zengerle.

Believe it or not, I managed to maintain a social life outside the PhD bubble. Thanks to all of my friends for supporting me, listening to my annoying complaints, and lifting me up again every time. A special thanks to Katharina Bina, Hanna Barysevich, and Alice Pairault. You all are amazing! Last, but not least, a big thanks goes out to my family.

Author's Publications

Large parts of the content of this dissertation have been published in journals or as part of the proceedings of peer-reviewed international conferences and workshops. In the following, we list all authored and co-authored publications of the author of this dissertation.

Main Publications

[Ged+17] Julien Gedeon, Christian Meurisch, Disha Bhat, Michael Stein, Lin Wang, and Max Mühlhäuser. "Router-based Brokering for Surrogate Discovery in Edge Computing". In: *Proc. of the International Conference on Distributed Computing Systems Workshops (ICDCS Workshops)*. 2017, pp. 145–150.

[Ged+18a] Julien Gedeon, Jens Heuschkel, Lin Wang, and Max Mühlhäuser. "Fog Computing: Current Research and Future Challenges". In: *Proc. of 1.GI/ITG KuVS Fachgespräche Fog Computing*. 2018, pp. 1–4.

[Ged+18b] Julien Gedeon, Nicolás Himmelmann, Patrick Felka, Fabian Herrlich, Michael Stein, and Max Mühlhäuser. "vStore: A Context-Aware Framework for Mobile Micro-Storage at the Edge". In: *Proc. of the International Conference on Mobile Computing, Applications and Services (MobiCASE)*. 2018, pp. 165–182.

[Ged+18c] Julien Gedeon, Jeff Krisztinkovics, Christian Meurisch, Michael Stein, Lin Wang, and Max Mühlhäuser. "A Multi-Cloudlet Infrastructure for Future Smart Cities: An Empirical Study". In: *Proc. of the 1st International Workshop on Edge Systems, Analytics and Networking (EdgeSys)*. ACM. 2018, pp. 19–24.

[Ged+18d] Julien Gedeon, Michael Stein, Jeff Krisztinkovics, Patrick Felka, Katharina Keller, Christian Meurisch, Lin Wang, and Max Mühlhäuser. "From Cell Towers to Smart Street Lamps: Placing Cloudlets on Existing Urban Infrastructures". In: *Proc. of the 2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE. 2018, pp. 187–202.

[Ged+18e] Julien Gedeon, Michael Stein, Lin Wang, and Max Mühlhäuser. "On Scalable In-Network Operator Placement for Edge Computing". In: *Proc. of the 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2018, pp. 1–9.

[Ged+19a] Julien Gedeon, Florian Brandherm, Rolf Egert, Tim Grube, and Max Mühlhäuser. "What the Fog? Edge Computing Revisited: Promises, Applications and Future Challenges". In: *IEEE Access* 7 (2019), pp. 152847–152878.

[Ged+19b] Julien Gedeon, Martin Wagner, Jens Heuschkel, Lin Wang, and Max Mühlhäuser. "A Microservice Store for Efficient Edge Offloading". In: *Proc. of the IEEE Global Communications Conference (GLOBECOM)*. 2019, pp. 1–6.

[Ged+20] Julien Gedeon, Sebastian Zengerle, Sebastian Alles, Florian Brandherm, and Max Mühlhäuser. "Sunstone: Navigating the Way Through the Fog". In: *Proc. of the International Conference on Fog and Edge Computing (ICFEC)*. 2020, to appear.

[Ged17] Julien Gedeon. “Edge Computing via Dynamic In-network Processing”. In: *International Conference on Networked Systems (Netsys’17): PhD Forum*. 2017, pp. 1–2.

[GS15] Julien Gedeon and Immanuel Schweizer. “Understanding Spatial and Temporal Coverage in Participatory Sensor Networks”. In: *Proc. of the 40th IEEE Local Computer Networks Conference Workshops (LCN Workshops)*. 2015, pp. 699–707.

Co-Authored Publications

[Heu+19] Jens Heuschkel, Philipp Thomasberger, Julien Gedeon, and Max Mühlhäuser. “VirtualStack: Green High Performance Network Protocol Processing Leveraging FPGAs”. In: *Proc. of the IEEE Global Communications Conference (GLOBECOM)*. 2019, pp. 1–6.

[Mar+19] Karola Marky, Andreas Weiß, Julien Gedeon, and Sebastian Günther. “Mastering Music Instruments through Technology in Solo Learning Sessions”. In: *Proc. of the 7th Workshop on Interacting with Smart Objects (SmartObjects ’19)*. 2019, pp. 1–6.

[Meu+17a] Christian Meurisch, Julien Gedeon, Artur Gogel, The An Binh Nguyen, Fabian Kaup, Florian Kohnhäuser, Lars Baumgärtner, Milan Schmittner, and Max Mühlhäuser. “Temporal Coverage Analysis of Router-Based Cloudlets Using Human Mobility Patterns”. In: *Proc. of the IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2017, pp. 1–6.

[Meu+17b] Christian Meurisch, Julien Gedeon, The An Binh Nguyen, Fabian Kaup, and Max Mühlhäuser. “Decision Support for Computational Offloading by Probing Unknown Services”. In: *Proc. of the 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2017, pp. 1–9.

[Meu+17c] Christian Meurisch, The An Binh Nguyen, Julien Gedeon, Florian Kohnhäuser, Milan Schmittner, Stefan Niemczyk, Stefan Wullkotte, and Max Mühlhäuser. “Upgrading Wireless Home Routers as Emergency Cloudlet and Secure DTN Communication Bridge”. In: *Proc. of the 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2017, pp. 1–2.

[Sch+12] Immanuel Schweizer, Christian Meurisch, Julien Gedeon, Roman Bärtl, and Max Mühlhäuser. “Noisemap: multi-tier incentive mechanisms for participative urban sensing”. In: *Proc. of the 3rd International Workshop on Sensing Applications on Mobile Phones*. PhoneSense ’12. ACM. 2012, 9:1–9:5.

[Wan+19] Lin Wang, Lei Jiao, Jun Li, Julien Gedeon, and Max Mühlhäuser. “MOERA: Mobility-agnostic Online Resource Allocation for Edge Computing”. In: *IEEE Transactions on Mobile Computing* 18.8 (2019), pp. 1843–1856.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

CHAPTER 1

Introduction

Chapter Outline

The present era of digitalization leads to a plethora of networked devices that capture, forward, and process vast amounts of data. For one part, these are personal devices, such as smartphones, smartwatches, on-body sensors, and head-mounted displays. The other part are small-scale sensors and actuators from the domain of the so-called *Internet of Things* (IoT). Recent studies by Cisco suggest that the number of devices connected to the Internet will reach 28.5 billion by 2020[1] and 500 billion by 2030[2]. It is often necessary or beneficial to carry out the processing of data outside those end devices, e.g., by performing *computation offloading*. There are three main reasons for this:

(i) The devices might have insufficient processing power to deliver satisfactory results in terms of quality of the computation result or the execution time. Albeit being equipped with powerful hardware, many devices remain inadequate for demanding tasks like video analytics. They also might lack specialized components that are indispensable for the task at hand, e.g., a GPU unit or FPGA.

(ii) Many of the devices are battery-powered. Form factor limitations and design requirements limit the size and, hence, the capacity of the battery. At the same time, battery life is a crucial factor for user satisfaction. Therefore, carrying out computationally intensive tasks that quickly drain the battery remains impractical.

[1] https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf (accessed: 2019-12-06)

[2] https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf (accessed: 2019-12-06)

(iii) Computations might be carried out *collaboratively* or require data originating from different sources. In case that transferring this data is inefficient, e.g., because of its size and/or the network connectivity, offloading the computation instead of the data provides a viable alternative.

For a long time, Cloud Computing was the predominant way to provide data processing services. Cloud Computing offers virtually unlimited resources that are concentrated in large data centers. Depending on the service model, providers of Cloud Computing infrastructure largely abstract away many operational and management problems from their users. Resources are virtualized and users can make use of a large pool of shared resources. Flexible pay-as-you-go models allow for seamless and economically viable scaling to one's needs. However, this service model also has some inherent drawbacks. Since Cloud Computing infrastructures are typically located far away from their clients (in some cases on another continent), there is sometimes a substantial end-to-end latency when accessing services hosted in the cloud. Furthermore, even today, bandwidth in core networks—those are transited when accessing cloud services—remains a scarce resource [Vul+15].

Until recently, these drawbacks of Cloud Computing had little practical impact. On the one hand, most data was both produced and consumed in the cloud, e.g., by applications in the domain of big data processing [Ji+12]. In such applications, data producers and consumers are predominantly well-connected clients, often located in datacenters themselves. On the other hand, applications for mobile devices were also carried out in the cloud because they were not latency-critical. This landscape has changed drastically today. New generations of personal devices (e.g., smartphones and augmented reality headsets) as well as small-scale sensors and actuators (e.g., from the IoT domain [Gub+13]) collect vast amounts of data at the edge of the network. With regards to the usage of this data, we can make the following observations:

(i) In many cases, the data is only relevant locally, i.e., it is not only gathered at the edge of the network but also consumed there.

(ii) The raw data and/or the result of processing is ephemeral, i.e., its temporal relevance is limited. Oftentimes, right after being transferred for processing, it can be discarded.

Prominent examples for applications whose data behave in such a way are real-time video analytics [Yi+17], cognitive assistance applications [Che+17b], mobile gaming [LS17], and autonomous driving [Lee+16]. For such applications, the drawbacks of Cloud Computing become more striking. For use cases with stringent requirements on the latency (e.g., to make real-time decisions in autonomous driving), Cloud Computing fails to deliver the required quality of service. In addition, today's core networks do not offer the bandwidth to support data transfers from the growing number of IoT devices and sensors.

These shortcomings of Cloud Computing in this changing landscape of devices and applications have led to the emergence of Edge Computing. Edge Computing is the concept of leveraging resources in close proximity to end devices—in a sense bringing the cloud closer to the edge of the network [Cha+14]. Tightly coupled with Edge Computing is the concept of *cloudlets* [Sat+09], micro data centers that offer proximate computing resources. Compared to the cloud, these resources are often leveraged opportunistically, i.e., users only make use of resources in their surroundings for a limited amount of time. Furthermore, resources are more heterogeneous, ranging from data center grade server hardware to single-board computers. Instead of relying on large, centralized resources in the cloud, the idea of Edge Computing is to make use of many decentralized resources in the vicinity of end devices. Typically, these resources are located in the access network, at the (wireless) gateway, or within 1-hop distance to the gateway, providing latencies in the range of single-digit milliseconds. Because many clients are highly mobile (e.g., a user's phone), they frequently need to migrate their data and computations between the resources at the edge—a stark contrast to rather static cloud deployments. Besides a reduced latency, processing data at the edge avoids using expensive links to distant cloud infrastructures. These characteristics make Edge Computing a crucial enabler for new classes of low-latency, high-bandwidth applications.

While some private, on-premise deployments of Edge Computing exist (e.g., in the context of *Industrial IoT*), today we are far from having *public* Edge Computing infrastructures available. This is especially true for urban environments. The vision of a *smart city* [Sch+11; Sch+16b] promises the usage of ICT[3] to provide liveable urban environments and cope with problems such as pollution or safety. Smart cities feature a dense network of highly interconnected mobile people and things. Future applications such as connected cars that drive autonomously and exchange data with devices in their surroundings are already being imagined. Such examples have strict requirements in terms of bandwidth and latency that make Edge Computing indispensable. However, we still lack essential building blocks for the realization of those applications. For one part, the requirements of individual types of applications are not well understood. Hence, the potential benefits of using Edge Computing remain unclear in many cases. Second, we lack the resources to carry out computations at the edge. While our urban spaces are filled with hardware that could host small-scale cloudlets, those are not made available today. Doubts remain about the practicability of having a dense, city-wide pool of computing resources to serve highly mobile users at any time. Third, the characteristics of Edge Computing differ from current Cloud Computing deployment models, and therefore, established mechanisms for the operation and management of infrastructure and applications cannot be applied to Edge Computing. Examples include the placement problem of data and computations, how to virtualize and share resources, and how to schedule access to those resources.

This thesis presents contributions in the field of *Urban Edge Computing* that close some of these gaps. Section 1.2 outlines our contributions in detail. The contributions provide answers to the following questions:

- What are the characteristics, benefits, and drawbacks of Edge Computing?
- What are the possible application domains for Edge Computing? Which of the benefits of Edge Computing are especially crucial for what types of applications?
- How and where can we provide ubiquitous computing resources in an urban environment?
- What is the granularity in which computations should be offloaded?
- Where should computations be carried out at the edge?
- How can edge resources be used in a (cost-)efficient way?

[3] Information and communications technology

- Can the edge be used as a distributed storage for user data?
- How do we need to adapt computing services in view of strict user requirements and constrained edge resources?

1.1 General Related Work

The contributions of this thesis are set in the field of Edge Computing. This section intends to present general related work that introduces Edge Computing and similar concepts to the unfamiliar reader. Related work specific to the individual contributions of this thesis is reviewed in sections 5.3, 6.2, 7.2, 8.2, 9.2, and 10.2.

The challenge to augment the capabilities of (mobile) devices by leveraging external resources has been envisioned for a long time in the field of *pervasive computing* [Sat01]. With the advent of Cloud Computing, such resources became widely available to realize this vision. Since then, *Mobile Cloud Computing* (MCC) [FLR13] has been the predominant way to offload computations.

An important conceptual enabler for the move towards Edge Computing was the introduction of *cloudlets* [Sat+09; Sat11]. In these initial publications outlining the concept, cloudlets have been defined as small-scale but resource-rich computing resources in the proximity of (mobile) users. They have been further described as a middle-tier between end devices and the cloud [Sat+14]. This notion of a middle-tier that is close to end users and devices led to the notion of Edge Computing [Sat17; Shi+16].

As we will further discuss in Chapter 2, other concepts exist, whose definitions sometimes lack a clear distinction from Edge Computing. Two of the most notable are *Fog Computing* and *Mobile Edge Computing* (MEC). Yi et al. [Yi+15; YLL15] define the concept of Fog Computing and use the term interchangeably with Edge Computing. Mahmud et al. [MKB18a] present a taxonomy of Fog Computing and define Fog Computing as an intermediate layer between IoT devices and the cloud. Mobile Edge Computing refers to the placement of computing resources in the radio access network [Bec+14; Mao+17; Abb+18]. Since this network is typically close to the end user, we see Mobile Edge Computing as *one* possible realization of Edge Computing.

Given the timeliness and broadness of the topic, various surveys explore the field of Edge Computing. Yousefpour et al. [You+19] present an extensive survey of publications related to Edge Computing and Fog Computing. Shi et al. [Shi+16] present several case studies and outline future challenges in Edge Computing. Similarly, Varghese et al. [Var+16] define motivations, opportunities, and challenges of Edge Computing. The survey of Li et al. [Li+18] focuses on architecture and management issues. Managing resources in an edge environment is a crucial building block. In this domain, Hong and Varghese [HV19] review publications and classify architectures, infrastructures, and algorithms for resource management in Edge Computing and Fog Computing. Further specialized surveys shed light on Edge Computing from the perspective of networking [Lua+16] or security [RLM18; YQL15; Sto+16].

No previous work provides a comprehensive overview and classification of applications that can benefit from Edge Computing. A comparison between cloudlets and Cloud Computing can be found in [Pan+15]; it is however limited to very few application scenarios. Existing works are either limited to a specific application domain, e.g., IoT [Has+18; Hec+18; Yu+18], or smart city applications [Per+17a; Tal+17]. Other classifications lack prominent examples like augmented reality [Pul+19]. We will close this gap with our application survey in Chapter 4.

1.2 Overview of Contributions and Thesis Structure

This thesis makes several contributions to the understanding, planning, deployment, and operation of a public Edge Computing infrastructure in an urban environment. An important characteristic of the Urban Edge Computing environment is its heterogeneity in two dimensions. First, several devices that capture and/or consume data are involved. For many of our experimental studies, we use mobile phones, however the results are applicable to other devices. These could range from small sensors and actuators to connected cars. Common to them is the need to perform computations or store data outside of the device. Second, we also consider the compute infrastructure to be heterogeneous, ranging from close-by equipment co-located with one's access gateway to resources in the transit network to the cloud. Contrary to previous works, we consider these heterogeneity characteristics to provide an integrated solution for Urban Edge Computing, ranging from the placement of physical resources to the adaptation of the runtime environment.

This thesis is structured into five parts. Figure 1.1 provides a complete overview of the structure of this thesis. The core contributions that provide building blocks for an Urban Edge Computing system are highlighted with a green background.

Part I contributes to the general understanding of the field of Edge Computing in two ways. First, we refine the definition of Edge Computing (Chapter 2) and analyze its characteristics, including advantages and drawbacks (Chapter 3). Second—based on the observation that it remains unclear which applications could benefit from certain aspects of Edge Computing—we present an extensive survey of application use cases in Chapter 4. We present a systematic way to break down applications by identifying four critical building blocks of applications (data consolidation, filtering & pre-processing, computation offloading, and data storage & retrieval). We then propose to classify applications according to the notion of *augmentation*, and for representative examples, we map the requirements of each application to how well they can be served by Edge Computing. The abstract notion of application building blocks allows us to generalize our findings and draw conclusions about the suitability of employing Edge Computing for certain types of applications. This part concludes with the definition of requirements for Urban Edge Computing that will be addressed in the remaining parts of the thesis.

Part II examines the *physical* infrastructure for Urban Edge Computing, i.e., which infrastructures can be leveraged in an urban environment to provide proximate computing resources. To this end, we suggest placing cloudlets on cellular base stations, WiFi routes, and augmented street lamps. Chapter 5 shows the viability of this idea by conducting a large-scale study of *urban cloudlet coverage* based on datasets captured in a major city. In Chapter 6, we present a placement strategy that decides which of those infrastructures to upgrade with cloudlets, taking into account their heterogeneity in terms of cost, communication range, and resources.

Part III presents an execution framework for Edge Computing that is based on the concept of composable microservices (Chapter 7). Unlike common offloading approaches, we advocate the concept of *store-based microservice onloading*, in which microservices are made available in a repository and do not need to be transferred from the client device, resulting in reduced end-to-end latency and energy consumption on the client device. Furthermore, we present a concept for the definition and execution of service chains, providing an easy way for developers of applications to use the framework. We implement the abovementioned concept in a prototype and evaluate it with applications running on a smartphone.

Part IV addresses runtime decisions of the previously presented execution framework. In particular, this concerns the problem of where to place functional parts of applications (Chapter 8) and data (Chapter 9). For the former, we present a heuristic-based placement strategy that minimizes the solving time with only a small optimality gap. For the latter, we present a middleware that makes rule-based storage decisions based on users' context. In Chapter 10, we extend our microservice-based execution to *adaptable* microservices. We introduce the concept of service adaptations in three different dimensions and showcase the tradeoff between execution time and quality of results that such adaptations can enable.

Part V concludes this thesis by summarizing its findings and giving an outlook on future work (Chapter 11).

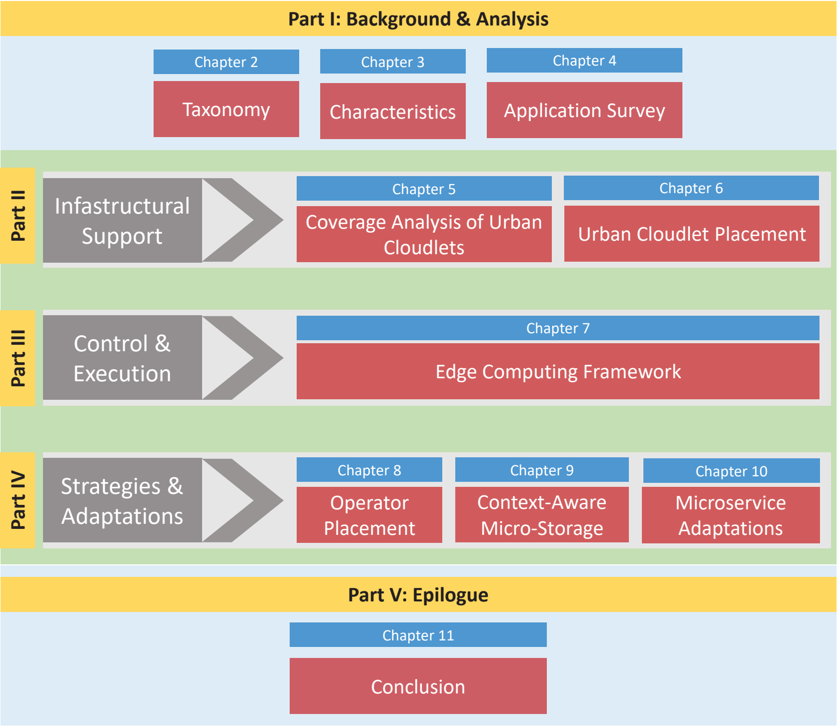


Figure 1.1: Overview of thesis structure

Part I

Background & Analysis

The first part of this thesis provides a detailed introduction and analysis of the field of Edge Computing.

Our first contributions consist of a taxonomy (Chapter 2) and analysis of Edge Computing characteristics (Chapter 3), thereby contributing to the general understanding of Edge Computing and the refinement of its definition.

Based on the identified characteristics, Chapter 4 performs a systematic survey of Edge Computing use cases, proposing a classification scheme and analyzing potential applications with regards to the previously defined characteristics of Edge Computing. Following the insights of this survey of the Edge Computing landscape, this part concludes with the definition of requirements for Urban Edge Computing.

This part is largely based on [Ged+19a]. Verbatim copies of text from this publication are printed in gray color throughout this part of the thesis. Tables and figures taken or adapted from this publication are marked with † in their caption.

CHAPTER 2

A Taxonomy of Edge Computing

Throughout the history of computing, a constant back-and-forth swinging between centralized and decentralized computing approaches has been observed [PA97]. We map this general observation to major milestones that have led to today's computing landscape, outlining four major eras in Figure 2.1. The first transition towards decentralized computing was the move from centralized mainframes to personal computers.

Cloud Computing. In the mid-2000s, we saw a major disruption with the advent of Cloud Computing. Cloud Computing offers abundant virtualized resources in large data centers. These resources can be used with flexible pricing models, often on a pay-as-you-go basis. Scaling in and out according to current demands can be done at a moment's notice and therefore removes the problem of over- or underprovisioning of resources.

We argue that Cloud Computing belongs to the category of centralized ap-

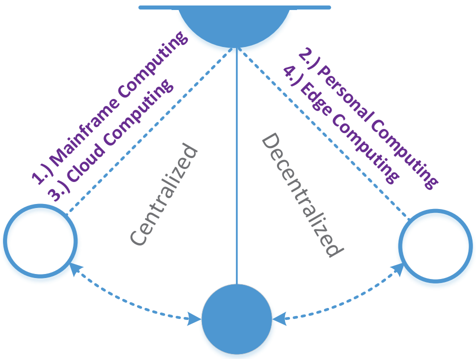


FIGURE 2.1: CENTRALIZED AND DECENTRALIZED COMPUTING PARADIGMS†

proaches because computing power is concentrated in a few distant locations (compared to the number of clients that use it). Leveraging cloud resources for offloading from mobile devices is termed *Mobile Cloud Computing* (MCC) [FLR13] for which a variety of frameworks exist [Cue+10; Chu+11; Kos+12; Kem+10]. Emerging classes of applications that require fast processing of large data generated from client devices and surrounding sensors have led to the latest distributed computing paradigm, termed *Edge Computing*.

Edge Computing. Edge Computing is the concept of placing and using storage and computing resources close to the (mobile) devices that produce and consume the data (see Definition 2.1). One contrasting approach is Cloud Computing, where said resources are located at data centers. Edge Computing is carried out using resources on *edge nodes* or *edge devices*. Similarly, the term *cloudlet* [Sat+09] has been coined to denote small-scale data centers close to users.

Entities in Edge Computing. In the context of computation offloading, nodes that are leveraged to perform computations are called *surrogates* (see Section 3.5.1). *Edge sites* are the physical environments where the edge resources are located. We refer to an *edge system* or *edge framework* to denote the entirety of resources at the edge, their clients, and control entities responsible for managing the resources.

Relationship between Edge Computing and Cloud Computing. It is important to note that Edge Computing aims not to be a replacement for Cloud Computing, but to complement it [Vil+16b]. This makes sense if we assume that every application needs access to three basic resources: (i) computation, (ii) communication, and (iii) storage. The need for computation and storage resources is well-served by Cloud Computing; in fact, the reason for the success of Cloud Computing is its capability to provide resource elasticity, which means that resources can be scaled in and out in order to instantly fit the customers' needs. On the downside, Cloud Computing cannot offer any guarantees w.r.t. the communication part because data centers are located away from the consumer, and typically, neither the cloud provider nor the user has full control over the transit network. Edge Computing solves this issue in the sense that it adds scalability in the network dimension, i.e., more users can be served with low-latency links when adding more edge sites (e.g., at users' wireless gateways). However, because of the limited resources at individual edge sites, Edge Computing cannot offer the same overall elasticity as Cloud Computing. Furthermore, realizing scalability in any of the three resource dimensions requires much more complex management in view of the dynamics in the network (e.g., caused by user mobility or sudden local changes in demands).

For these reasons, in practice, we expect an interplay of Cloud Computing and Edge Computing. Edge Computing offers the additional scalability required for processing locally relevant tasks in an environment with a large number of data generators and consumers. Complex, long-running, and data-driven tasks that are not time-critical will benefit more from the abundance of scalable resources in the cloud. Similarly, Edge Computing will most likely not be able to replace the cloud for the long-term storage of data because of the limited capabilities of edge devices. However, user-facing time-critical tasks may benefit from a reduced latency in the critical path when using infrastructure at the edge, and this might also include the caching of ephemeral data on edge devices. In addition to the latency benefit, caching data at the edge has the potential to reduce bandwidth usage in the core network.

Differences in resources and their distribution. While Cloud Computing offers virtually unlimited resources in geo-distributed data centers, resources in Edge Computing are locally clustered around its consumers. These computing nodes are more heterogeneous w.r.t. to their available resources and are often leveraged opportunistically. Especially if we consider non-redundant, consumer-grade devices for computations, the availability and reliability of resources at the edge might be limited. However, Edge Computing can make the communication more reliable, in the sense that it can offer an alternative if network links to the cloud break down. This is especially interesting for disaster scenarios where Edge Computing can offer an alternative infrastructure to keep critical tasks alive. Edge resources can often be accessed within one hop from the wireless gateway that users are connected to. Ideally, Edge Computing systems can support user mobility, e.g., by migrating user data and computations to the next proximate location. In addition, wireless gateways can provide additional contextual information to the application, something not available in Cloud Computing.

Computation offloading and virtualization. Another major difference is the granularity of offloading. In Cloud Computing, we see large parts of applications being moved to remote resources, while at the edge, offloading is more fine-grained and needs a more careful decision of what to offload. Individually offloaded components at the edge are often part of a processing pipeline consisting of several of those components that do not necessarily run on the same edge device. Additionally, we can observe that because of limited resources at the edge and the higher user dynamics, virtualization technologies used for Edge Computing tend to be more lightweight. For example, containers are often used instead of virtual machines (see Section 3.5.2).

Loose coupling. We define the loose coupling between clients and the computing and communication infrastructure as another important characteristic of Edge Computing. This is an especially important characteristic in Urban Edge Computing, where the computing resources are shared among multiple users and applications. Other concepts that are sometimes referred to as Edge Computing deploy static resources on-premise for one particular user and application. Those deployments, however, do not face the same challenges, e.g., with regards to network coverage and connectivity, device and data mobiltiy, or scaling.

Summary. To conclude this section, Table 2.1 summarizes the differences between Edge Computing and Cloud Computing.

2.1 Terminology

For the remainder of this thesis, we will use the terms *Edge Computing* and *Urban Edge Computing* as described in Definition 2.1. Besides the term *Edge Computing*, other terminology that denotes similar concepts exist, most notably the term *Fog Computing* [Yi+15; Bon+12; Ged+18a]. Fog Computing is a term originally coined

TABLE 2.1: COMPARISON OF CLOUD COMPUTING WITH EDGE COMPUTING †

	Cloud Computing	**Edge Computing**
Proximity to client devices	low	high
End-to-end latency	high	low
Infrastructure	centralized data centers	decentralized cloudlets
Heterogeneity of computing hardware	low	high
Number of computing resource locations	few	many
Resources at individual locations	many	few
Geo-distribution of computing resources	locally clustered	widespread
Availability & reliability of resources	high	varying
Virtualization	heavyweight	lightweight
Connection to resources	long-thin	short-fat
Access to resources	through core network	typically via 1-hop wireless gateway
Applications	data-driven	user-driven
Offloading granularity	mostly entire applications	computationally intensive and latency-critical parts

by Cisco [Bon+12] in the context of their *IOx* platform, envisioning to leverage untapped processing power in network middleboxes when those are either overprovisioned or not running at full load.

While the terms *edge* and *fog* both allude to the same concept—processing data close to end devices—it is worth noticing that there is a broad spectrum of (sometimes blurry) definitions and arguments in trying to define the differences between the two. One possible distinction is that Fog Computing extends the cloud towards the edge, while Edge Computing originates from the need of end devices to offload computations. However, the exact definitions remain an ongoing discussion in academia [Mar+17; VR14]. Closely tied to the concept of Fog Computing are *Cloudlets*. Cloudlets have been described as a middle-tier [Sat+14] that extends the cloud towards the edge [Ver+12a; Lew+14].

Mobile Edge Computing (MEC)—more recently termed Multi-Access Edge Computing—refers to the colocation of resources at the Radio Access Networks (RAN), e.g., at cellular base stations [Abb+18]. This can therefore be considered as a special case of Edge Computing, mostly from the point of view of mobile network operators. Especially with the advent of the fifth generation of cellular networks (*5G*), MEC deployments are expected to gain more importance [Hu+15b; Nun+15].

Other hybrid terms exist, notably *Mist Computing* [Pre+15] and *Osmotic Computing* [Vil+16b]. The former can be thought of being similar to Fog Computing but closer to the edge devices, while the latter advocates a seamless migration of services from data centers to the edge.

> DEFINITION 2.1: EDGE COMPUTING AND URBAN EDGE COMPUTING
>
> *Edge Computing* denotes the general concept of placing computing and/or communication resources close to the action scene, e.g., in proximity of users, sensors, or actuators. We refer to *Urban Edge Computing* as the application of this concept in an urban environment.

CHAPTER 3

Characteristics of Edge Computing

Chapter Outline

In this chapter, we analyze the characteristics of Edge Computing. We start by analyzing what the potential benefits and drawbacks of Edge Computing are (Section 3.1). We then discuss the characteristics of "the edge" in terms of communication (Section 3.2), the involved devices (Section 3.3), and stakeholders (Section 3.4). Lastly, we review enabling technologies for Edge Computing in Section 3.5.

3.1 Promises, Benefits, and Drawbacks

As outlined in the previous chapter on the taxonomy of Edge Computing, the general idea is to move storage and processing capabilities from the cloud closer to the clients and towards the origin of the data, often by opportunistically using the infrastructure in a highly dynamic mobile environment. This potentially brings a number of advantages, the most important of which we describe in the following:

Advantage I: Lower latency | As we will discuss in Chapter 4, many types of applications have stringent requirements on the end-to-end latency, i.e., the overall time from requesting a service (e.g., a computation) to obtaining the result. One important factor on this critical path is the network delay. Cloud Computing infrastructures are geographically widely distributed across data centers, and the user typically has little to no control over where the requests will be processed. Hence, it is not uncommon for requests to be directed to distant data centers.

Many works have presented empirical measurements of network latencies and motivated Edge Computing based on those numbers [Che+17b; Ged+17; Sat+09]. For instance, in [Sat+09] the authors measure the mean network round-trip times between New York and Berkeley to be 85 ms. If we now imagine an application that needs to process a video scene in near real-time with a delay constraint of less than 50 ms, the mere network latency already violates this constraint. Even by optimizing transit networks, physical lower bounds remain. In contrast, the access delay to a nearby wireless gateway in the case of WiFi is typically in the order of magnitude of a few milliseconds. Chen et al. [Che+17b] have conducted extensive empirical studies using a cognitive assistance application and conclude that using the cloud over a cloudlet adds around 100–200 ms of latency. Besides the physical lower bounds of transmissions, the *network jitter*, i.e., the variation in delay is another issue for latency-critical Edge Computing applications. This variance is caused, e.g., by different load levels in the network and makes *guaranteeing* a latency close to the lower bound impossible.

Advantage II: Less bandwidth utilization in the core network | In the current landscape of billions of mobile devices that generate data, we observe that captured data often only is of limited spatial and temporal relevance. As an example, we can imagine an intelligent scheduling scheme for traffic lights that is based on reported sensor data from vehicles [BB14]. In this example, the data is relevant only for the time the vehicles are in the vicinity of the traffic light. Applications often do not consume every individual sensor reading, but data that is derived from those individual readings, e.g., aggregate or filtered values, inferred events, or outliers. If, however, all raw values would be streamed to the cloud for analysis, this might overload the core network. This is especially relevant since wide-area network bandwidth remains a scarce resource [Vul+15]. The same holds true for many of today's wireless access networks, e.g., as motivated in [Wan+18b]. Especially large, continuous data streams can be a burden on backhaul networks. Distributed processing and aggregation of data streams along the path to the consumer can help to mitigate this. In the domain of Wireless Sensor Networks this is a popular approach [Fas+07] that can easily be mapped to aggregation by intermediate edge nodes.

Besides aggregation, Edge Computing can also offer storage capabilities [May+17] that take into account contextual information for the decision on where to store the data [Ged+18b]. For example, at large-scale events with overloaded mobile networks, edge nodes can provide storage to share data among people that are close-by. Chapter 9 will present our contribution of a context-aware edge storage framework. Other works have investigated edge storage for caching [Zha+15a] or buffering of IoT data [Psa+18]. It is worth noticing that most of these works assume the data to be short-lived. However, storing non-ephemeral data on unreliable edge nodes requires replication mechanisms, as demonstrated in [MRS19]. The savings in data transfers to the cloud when using Edge Computing has been demonstrated in practice with various use cases, from document synchronization [Hao+17] to mobile gaming [Var+17]. For example, Hao et al. [Hao+17] demonstrate a reduction in data transfers to the cloud of up to 90% in an application for document synchronization.

ADVANTAGE III: ENERGY SAVINGS AND INCREASED ENERGY EFFICIENCY | Mobile devices have an inherently limited battery life. Advances in battery technology have not kept pace with the increased processing capabilities of modern mobile devices [AS13; KAB13]. Furthermore, their small form factors limit the size of the battery. Battery life is an important factor for the overall user satisfaction [Hav11] and remains an important constraint for many applications, e.g., mobile gaming [Hua+14]. Carrying out compute-intensive tasks on the device is detrimental to the device's battery life and, thus, has a negative impact on the user's experience. This factor is even more crucial for small-scale sensors that are deployed in the environment and designed to never be serviced. In this case, the battery life equals the lifetime of the device. Therefore, moving the computations away from the devices is beneficial for their battery life. This has been shown for both cloud [KL10] and edge [Hu+16] infrastructures.

Saving energy is not only important for end devices, but also for edge nodes on which the computations take place [RAD18]. Besides the advantage of reduced operational cost, Edge Computing nodes are often enclosed in tight physical spaces and therefore, heat dissipation must be limited. Hence, many works have presented energy-efficient mechanisms for resource allocation [You+17], offloading [Nan+17; Zha+18d], and data delivery [Jay+14] in Edge Computing. Xiao et al. [XK17] suggest cooperative offloading, in which edge nodes forward tasks among each other. The authors study the tradeoff between quality of experience for users and the fog nodes' energy efficiency and present a cooperation strategy for optimal workload allocation.

The previous examples have outlined the partial benefit from the point of view of mobile devices and edge nodes. However, it is important to note that to analyze the overall energy benefit of Edge Computing, we need a more holistic view. While offloading might save battery life on the mobile device, this does not answer the question of whether the chosen edge resources are more energy-efficient compared to Cloud Computing infrastructures. As one approach, Jalali et al. [Jal+16] take into account the energy efficiency of the access network that is used when performing Edge Computing. The authors conclude that micro data centers at the edge can indeed be more energy efficient than Cloud Computing. They further identify the processing of continuous data streams as an ideal edge application, especially when those data streams are on end user premises and have a low access rate (e.g., video surveillance). Boukerche et al. [BGG19] survey energy-efficient offloading in Mobile Cloud Computing from the perspective of both the mobile device and the cloud infrastructure. The authors consider different types of deployments and especially mention the possible energy overhead of the offloading process.

ADVANTAGE IV: BETTER PRIVACY AND DATA PROTECTION | In Cloud Computing, users typically have little control over their data and where exactly it is processed. Yet, users' end devices generate more and more data at the edge, many of which is personalized and privacy-sensitive, e.g., in healthcare-related applications. As users become more sensitive to privacy issues, they might not be willing to accept the current practice of how data is processed. For example, Davies et al. [Dav+16] outline how privacy concerns hinder user acceptance of IoT deployments.

Edge Computing offers the opportunity to act as a privacy-enabling mediator between the user's data and cloud-based services, especially when users have access to edge infrastructures that are within their trust domain or that are operated by trusted providers. Since Edge Computing resources are offered on-site, providers are subject to local laws and regulations. Depending on the location, this can give certain assurances, e.g., with regards to data protection laws. Furthermore, following the same argument, Edge Computing providers can typically be held accountable more easily than (foreign) Cloud Computing providers. Edge Computing allows the application of privacy-preserving mechanisms (e.g., as proposed in [Shi+11]) early in the processing chain and close to the data source, hence reducing the impact of potentially untrustworthy processing entities that subsequently handle the data. The fact that in a public Edge Computing infrastructure, multiple providers would be involved further strengthen the benefits with regards to privacy and anonymity. Protocols like *cMix* [Cha+16] have shown that anonymous communication can only be broken if all parties cooperate to do so. We argue that this is an unrealistic scenario in a multi-provider Edge Computing infrastructure.

Besides data from individuals, data collected in public spaces is also relevant to privacy. For example, a camera mounted on top of a road intersection captures video streams that are used to optimize traffic and dynamically adapt the traffic lights. This application may be realized in different processing steps, e.g., detecting cars in individual lanes, aggregating their number, computing a strategy to optimize the traffic, and so forth. To preserve drivers' privacy, the blurring of license plates would be a critical task that has to be carried out at the edge before transferring the video streams for further analysis. Similarly, Basudan et al. [BLS17] present an encryption scheme to ensure privacy when monitoring road conditions. Other works explore privacy-preserving publish-subscribe mechanisms at the edge [Wan+17] or how edge infrastructures can help in the dissemination of information containing certificate revocations [Alr+17].

While many of these potential benefits are acknowledged in literature, less attention has been directed to the possible drawbacks of Edge Computing. In particular, we consider the following aspects to be problematic:

DRAWBACK I: UNRELIABLE DEVICES | Because Edge Computing relies on small-scale, often consumer-grade devices that are used opportunistically as edge nodes, their reliability cannot compete with advanced measures for reliability in data center environments, such as UPS[1], emergency power systems, redundant cooling, redundant network connections and high-speed interconnections that enable large-scale replication. Edge Computing must therefore either be tolerant of failures or mitigate the effects via replication schemes, e.g., by replicating stored data across edge nodes [MRS19].

DRAWBACK II: LOW INDIVIDUAL COMPUTING POWER | The computing power of individual edge nodes is usually much lower compared to a cloud data center.

[1] uninterruptible power supply

For latency-tolerant heavy computations, such as neural network training, the cloud will remain the predominant deployment model.

Drawback III: Limited scale-out capabilities | Since the capacity at each edge site is limited, it is much more difficult to scale out edge applications with high demands in a small area. Because data centers are designed to serve large geographical areas, local spikes in demand, e.g., during an event, are small in comparison to the total demand. In contrast, Edge Computing infrastructure may be overwhelmed in such a situation as the area over which extra demand can be distributed while still fulfilling good quality of service might be limited.

Drawback IV: High operational expenses | Edge Computing is likely to be more expensive than traditional Cloud Computing, which benefits much more from economies of scale. Cost benefits of large-scale data centers [Gre+08] that cannot be exploited in Edge Computing include rental cost, energy cost, and personnel cost. Data centers are often built in places with low taxes, low land costs, and low energy prices. They concentrate vast amounts of homogeneous servers and networking hardware in one easy to reach location. Cloudlets, however, must be geographically much closer to their clients, which prevents strategic positioning in low-price areas. Also, due to the distribution over many small-scale locations, the maintenance is much more complex. For Edge Computing to be economically viable, the resulting higher cost must be compensated by lower data transmission costs or other benefits, like increased privacy or the need for ultra-low latency.

Drawback V: Concerns about security and trust | The idea to opportunistically leverage devices in one's surroundings to carry out computations and store data naturally raises concerns about such a system's security and trustworthiness. According to [Muk+17], existing mechanisms for the cloud cannot be applied to edge environments. Roman et al. [RLM18] survey the security threats and corresponding challenges in Edge Computing and Fog Computing. To make Edge Computing pervasive, unified trust models and authentication mechanisms across stakeholder boundaries are required.

3.2 Access Technologies and Communication Patterns

We now turn our attention to the typical access technologies and communication patterns in Edge Computing. While Cloud Computing is accessed through wired backhaul connections, one characteristic of Edge Computing is that clients are typically connected to the edge resources through wireless gateways. We expect edge resources to be either colocated on those gateways or within 1-hop distance. To connect to wireless gateways, client devices use different access technologies. The most common are WiFi [Ged+17] or cellular [SBD18] connections. In this domain, we can observe development in two aspects:

(i) New communication standards emerge | One example of emerging standards are 5G networks that are expected to be deployed soon. 5G not only promises much higher bandwidth and lower latencies compared to current cellular networks, but it will also provide additional services like context-awareness on the network access layer [Ban+14]. Other future wireless access technologies include millimeter wave [Aba+] and visible light communication [JLR13]. These (future) technologies will contribute to a widespread coverage of high-speed wireless access points.

(ii) Novel gateway devices | Existing access technologies will be embedded into new devices that can act as gateways. One example in the urban space are street lamps, as we will describe in Chapter 5. While today only providing lighting, emerging *smart* lamp posts are designed to offer colocated access and computing resources. A second example is to leverage computing resources present on modern cars [Hou+16]. It has also been suggested to place computing resources on UAVs[2] [Sat+16; JSK18b].

Besides the wireless access technologies, we can also distinguish Edge Computing systems by their communication patterns. Generally speaking, our computing world consists of humans and various things that are connected. Depending on which entities communicate with whom, different terminology is used, such as Machine-to-Machine (M2M), Device-to-Device (D2D), Car-to-Infrastructure (C2I), Car-to-Car (C2C), etc. The important distinction between all those terms is whether we have autonomous communication between devices or human actors in the loop.

3.3 Device Ecosystem

One of the most striking characteristics of Edge Computing is the heterogeneity of devices that are involved in the capture and processing of data. The number of mobile devices and sensors that capture data has dramatically increased in recent years. One prominent example are today's smartphones. According to a recent study[3], the number of mobile broadband subscriptions has reached 6 billion as of today and will grow to over 8 billion by 2024. Similarly, the IoT aims to connect a variety of objects such that these are able to communicate with each other [AIM10]. Other end devices include smartwatches, smart glasses, and personal on-body sensors. Common to all of them is that they generate large amounts of data that need further processing to provide additional services. This need for processing has been one of the most important aspects for the development of Edge Computing.

In Edge Computing, not only end devices are heterogeneous, but also the devices on which the computations are carried out. Every device in the vicinity of the mobile client that has spare resources to perform computations can be considered for Edge Computing. This can range from consumer-grade hardware to hardware designed for data centers. For example, small-scale single-board computers such as Raspberry Pis have been used in Edge Computing [Pah+16; BZ17] as well as home routers [Meu+15] or compact setups with more powerful hardware [RAD18]. In between end-user locations and cloud data centers, we can also leverage network middleboxes that have additional computing capacity available. This was the initial use case for Cisco's vision of Fog Computing [Bon+12]. As we move closer to the edge of the network, we expect to find devices with fewer resources but in greater

[2] unmanned aerial vehicle

[3] https://www.ericsson.com/en/mobility-report/reports/june-2019 (accessed: 2020-03-22)

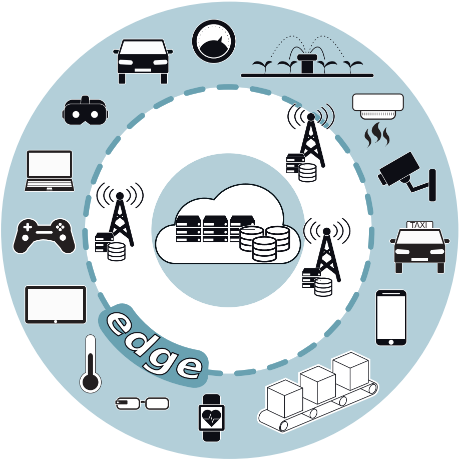


FIGURE 3.1: HETEROGENEOUS EDGE COMPUTING DEVICE ECOSYSTEM[†]

number. This reverses as we move closer to the cloud because most locations at the edge of the network cannot physically accommodate the resources found in a data center.

In summary, the ecosystem of devices is very heterogeneous, both in terms of their functions and form factors, but also regarding their capabilities and computing power. Figure 3.1 depicts this heterogeneous ecosystem of devices that are involved in Edge Computing, ranging from sensors and consumer devices to more powerful computing infrastructure as we move towards the core network. The heterogeneity in Edge Computing is a main requirement that planning and runtime decisions (e.g., with regards to the placement and assignment of resources) need to consider.

3.4 Stakeholders and Business Models

In the previous sections, we discussed Edge Computing's heterogeneity w.r.t. applications and devices. The variety of stakeholders is another dimension of heterogeneity in Edge Computing. Stakeholders in this context are individual users or organizations that (i) use services deployed at the edge, (ii) operate edge infrastructure, or (iii) benefit from edge deployments. Figure 3.2 shows an example of three stakeholders (the *user* of a service, the *service provider*, and the *infrastructure provider*). The figure also shows their (partially overlapping) interests.

Users have certain requirements and expectations regarding the quality of service delivered by applications. Notable examples are a low (perceived) latency and high service availability. For many future use cases, leveraging Edge Computing to meet those demands will be indispensable. Service providers in turn are responsible for ensuring that their services can meet these demands to satisfy their customers. For infrastructure providers (e.g., ISPs and operators of cellular networks) Edge Computing can be an opportunity to generate additional revenue. This, for example, can be achieved by either offering (virtualized) resources at the access network or renting out space for server colocation at those access gateways. Besides commercial offerings, we can also imagine that Edge Computing infrastructures will be offered free of charge. For example, cities could provide those as a service to their

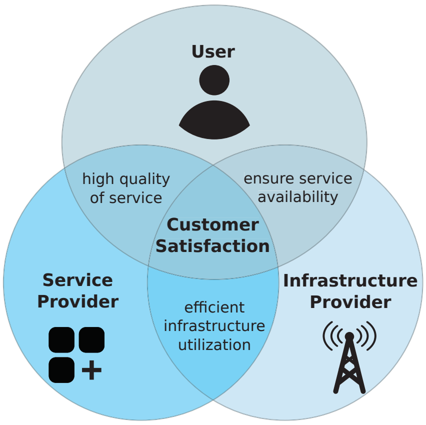


FIGURE 3.2: STAKEHOLDERS IN EDGE COMPUTING †

citizens [Mot+13]. Similarly, private citizens could offer resources for free. This kind of sharing economy has already been seen for providing WiFi connectivity [EFP10]. Therefore, offering computing power could be a next step.

The diverse ownership of edge resources and the lack of a unified business model is also one of the major reasons why finding appropriate business models for Edge Computing remains a challenge (see Section 11.2.3).

3.5 Enabling Technologies

A number of enabling technologies have lately fostered the development of Edge Computing. For example, encapsulating and moving parts of an application would be difficult without lightweight virtualization standards. The most important of those enabling technologies are summarized in this section.

3.5.1 Offloading Mechanisms

Computation offloading—sometimes also named cyber foraging [Bal+02; BF17]—is the process of running an application or parts thereof outside the client device [Kum+13; MB17] on a so-called *surrogate*. Surrogates are computing resources that execute tasks on a client's behalf [GC04]. The motivation for computation offloading can be twofold. The first reason is related to the limited resources of mobile devices. Either the device does not have enough resources, or the resources would only be able to produce an inaccurate or unsatisfiable result. In many cases, offloading to powerful surrogates can reduce the execution time [KYK12] of the task. The other benefit of offloading is related to energy considerations, as pointed out in Section 3.1. In this regard, computation offloading can help to save energy on a mobile device and hence prolong its battery life, as demonstrated in [KL10; LWX01]. Furthermore, the parallel execution of offloaded tasks can greatly improve the system's scalability [Kos+12]. Kumar et al. [Kum+13] and Sharifi et al. [SKK12] provide extensive surveys on computation offloading.

Offloading decisions. To perform offloading, it needs to be decided *what* to offload, *when* to offload and *where* to offload to. What to offload is typically determined by a component—the application profiler—that automatically partitions the applications into offloadable and non-offloadable parts [Cue+10; Chu+11]. The parts that are to be offloaded can also be specified manually, e.g., through annotations made by the developer. When and where to offload is decided by scheduling mechanisms, e.g., as presented in [DH15]. It is worth mentioning that deciding if something should be offloaded requires taking the overhead into account. Offloading typically also means that the logic and in some cases the execution environment has to be transferred to the surrogates, which in turn consumes energy and incurs additional latency. This is especially true for low-quality wireless connections. Therefore, the offloading decision should be based on a careful tradeoff and ideally take contextual information (e.g., remaining battery, network conditions, and QoS requirements) into account [Zho+17]. This is a well-studied optimization problem. As an example, the work of Chen et al. [Che+16] focuses on the offloading decision itself. Miettinen and Nurminen [MN10] analyze the critical factors for energy-efficient offloading. Many applications [Alt+16; Azi+17; Zha+17] take into account the different characteristics of edge and cloud and hence, offload delay-tolerant tasks to the cloud and time-critical tasks to the edge.

Offloading granularities and offloading targets. Offloading can be done in different granularities, e.g., entire virtual machines [Shi+13], threads [Gor+12], or pieces of code [Cue+10]. Most of the existing approaches are aimed at MCC, i.e., the surrogates are located in the cloud. One notable exception is *Paradrop* [LWB16], a platform for the deployment and orchestration of containerized applications on WiFi access points. Golkarifard et al. [Gol+19] present a framework that supports both cloud and edge offloading for wearable computing applications. As an alternative approach for offloading, we will present the concept of a *microservice store* in Chapter 7.

3.5.2 Lightweight Virtualization

Offloading computations requires the packaging of application logic and/or execution environments into units that can be re-used across runtime instances. Computing resources in Edge Computing typically feature multi-tenancy, i.e., they are shared among multiple applications. Therefore, offloaded applications typically run in virtualized environments. Virtualization serves two main purposes: flexibility and isolation. However, virtualization entails a tradeoff that needs to be balanced carefully in Edge Computing environments. On the one hand, virtualized environments should have little overhead in terms of management complexity and startup time when comparing them to processes running on a shared operating system. On the other hand, given the multi-tenant nature of virtualized environments, security and privacy considerations dictate strong isolation between different applications on the same edge device. The type of virtualization environment furthermore determines which techniques for application migration can be used [Pul+18].

Virtual machines. The seminal paper that introduced cloudlets [Sat+09] envisioned virtual machines (VMs) as the virtualization layer. However, virtual machines encapsulate entire operating systems, and while they provide good isolation, they incur a large overhead in terms of size and startup time. Hence, in practice, two viable virtualization technologies have emerged for Edge Computing [Mor+18a]: *containers* and *unikernels*.

Container-based virtualization. Containers do not need a separate guest operating system for each application. Instead, this technology uses virtualization on the operating system level. The operating system, its kernel, and libraries are shared and, hence, the isolation is weaker compared to VMs. However, this weaker isolation is traded for enhanced performance, e.g., by a lower startup time and smaller size of application images. Ramalho and Neto [RN16] and Felter et al. [Fel+15] have analyzed the performance difference between containers and VMs in detail, outlining, e.g., the superior performance of containers with regards to disk and network throughput. Using containers also simplifies the management of hardware resources, since only one OS has to be maintained. A container engine provides a format for bundling applications and an interface to control the execution of containerized applications. An important feature of container-based virtualization is that components can be reused across different containers. Often, a so-called base image is used, on top of which additional components and dependencies are loaded. Consequently, the image of a containerized application is rather small, as both the base image and additional dependencies are typically loaded upon the container's invocation. This allows shipping applications as smaller, portable units compared to virtual machines. These are desirable properties that make containers a viable virtualization technology for Edge Computing [PL15].

Besides access to virtualized resources like CPU or memory, the container engine can also provide applications with network connectivity, both internally as well as mapped to the operating system's ports. From a technical perspective, Linux-based container engines use two main features of the kernel to realize containerized applications: *cgroups* to control resource utilization and *namespaces* to provide isolation. In practice, Docker[4] has emerged as the de-facto standard container platform, although others exist, e.g., LXC/LXD[5], OpenVz[6], Rocket[7], and podman[8]. One advantage of Docker is the easy syntax of the *Dockerfile* through which container images are defined. In addition, powerful tools for the orchestration of Docker containers exist, most notably Docker Swarm[9] and Kubernetes[10]. Consequently, Docker has been used in various approaches for implementing Edge Computing prototypes, e.g., [LWB16; BZ17].

Unikernels. While recent efforts have been directed towards shrinking the size of containers [Tha+18], unikernels are an even more lightweight approach to virtualization than containers [Mad+13; MS13]. Contrary to containers, which share all of the operating system's libraries, unikernels can be considered an isolated bootable image that can run on bare metal or a type-1 hypervisor. The difference to virtual machine images is that unikernels do not contain a complete operating system and its libraries. Instead, only the parts required to execute the

[4] https://www.docker.com/ (accessed: 2019-08-09)
[5] https://linuxcontainers.org/ (accessed: 2019-08-09)
[6] https://openvz.org/ (accessed: 2019-08-09)
[7] https://coreos.com/rkt/ (accessed: 2019-08-09)
[8] https://podman.io/ (accessed: 2019-08-09)
[9] https://docker.com/products/orchestration (accessed: 2019-08-09)
[10] https://kubernetes.io/ (accessed: 2019-08-09)

functionality are included, hence they are often referred to as *library operating systems*. Everything required to run the application is compiled into the unikernel's image. At runtime, unikernels run as a single-process and have no distinction between user space and kernel space. This architecture leads to a very fast boot time and execution speed of unikernels, since many management functionalities such as context switching, scheduling, and the management of virtual memory are non-existent. Furthermore, their reduced attack surface (since no unused components and libraries are included) makes them more secure. Due to their restrictions, e.g., no forking is supported, not every application can immediately be packed into a unikernel. Some projects like Unik[11] aim to automate unikernel compilation and deployment, but in general, this is highly specific to the particular unikernel, and a unified orchestration of unikernels remains challenging. The unikernel landscape is rapidly evolving, with new projects constantly emerging. At the time of writing, MirageOS[12], Rumprun[13], OSv[14], ClickOS[15], and HalVM[16] are among the most popular and active unikernel projects. Some works have already applied unikernels to Edge Computing environments. Cozzolino et al. [CDO17] have suggested unikernels for edge offloading. Wu et al. [Wu+18] advocate the concept of a rich unikernel to support various applications. As a proof-of-concept, the authors integrated Android system libraries into a OSv unikernel.

3.5.3 Software-Defined Networking

Software-Defined Networking (SDN) has emerged from the paradigm of *active networking* [BCZ97]. The basic idea of SDN is to separate the control plane (i.e., the part that determines the behavior of network functions) from the data plane (i.e., the entities that forward data) of the network [Kre+15]. SDN brings advantages from the management and operation's perspective of computer networks. For example, the management of networks is simplified because devices do not have to be configured independently and manually. Instead, configurations are performed through a (logically) centralized control entity. Users can specify rules in a high-level way, which are then translated by the controller and applied to network devices via protocols like *OpenFlow*. These protocols can be used to perform management functionalities in the network, such as defining flows or network slices.

Future potential of SDN. While SDN today is mainly used to control the forwarding of data, future implementations of the general concept can be a crucial enabler for Edge Computing because of two main reasons. First, SDN abstracts away complex management tasks from the user. For example, the control plane could be responsible for the placement and orchestration of services. Users would just specify what service they request, and the decision where to instantiate it would be taken care of by policies at the control plane. Similarly, the controller's global view can be leveraged for service discovery and to collect measurement data on the state of the network. Second, the capability of SDN to dynamically reconfigure the network is crucial in dynamic edge environments. For instance, these dynamics are related

[11] https://github.com/solo-io/unik (accessed: 2019-08-09)
[12] https://mirage.io/ (accessed: 2019-08-09)
[13] https://github.com/rumpkernel/rumprun (accessed: 2019-08-09)
[14] http://osv.io/ (accessed: 2019-08-09)
[15] http://cnp.neclab.eu/projects/clickos/ (accessed: 2019-08-09)
[16] https://github.com/GaloisInc/HaLVM (accessed: 2019-08-09)

to user mobility or changes in service demands. In case of necessary migrations, e.g., due to intermittent connectivity to unreliable compute nodes, SDN-enabled networks can push new flow rules to the network in order to redirect traffic. Furthermore, SDN can also help to provide guarantees on the quality of service delivered by edge services, e.g., by reserving bandwidth on network links. Few works have already applied some principles of SDN to edge environments. For example, Heuschkel et al. [Heu+17] present a protocol to extend software-defined control beyond the core network to the end devices. Bi et al. [Bi+18] show how user mobility can be realized by decoupling mobility control and data forwarding through SDN. An extensive overview of how Edge Computing can benefit from SDN can be found in [BOE17].

3.5.4 Network Function Virtualization

Network Function Virtualization (NFV) is the concept of decoupling network functions from the dedicated hardware appliances they are typically deployed on. Instead, the concept of NFV aims at deploying those functions as software components atop of virtualized infrastructures. Examples for such network functions include deep packet inspection (DPI), firewalls, software-defined radios (SDR), and network address translation (NAT), among others. Because all the functions are built in software and run on virtualized hardware, making changes to them is very fast and easy. One popular example of an NFV platform is ClickOS [Mar+14]. A comprehensive survey about the current state of NFV can be found in [Mij+16].

Future potential of NFV. The main benefit of NFV for network operators and service providers is to make the deployment and operation of their network more cost-efficient [Haw+14]. Moving towards virtualized network functions is also interesting in view of new network technologies. For example, Abdelwahab et al. [Abd+16] show how NFV can help to fulfill the requirements of 5G networks. Hence, there is a big interest in NFV as a business model.

At the same time, we can observe a kind of "chicken-or-egg problem" when asking the question of why no widespread edge infrastructure is available yet, e.g., at the radio access network (RAN). Taking the example of a RAN, a network operator currently might not see a business opportunity for Edge Computing because very few novel applications that would benefit from it (and users willing to pay to use the service) exist. Similarly, the lack of an Edge Computing infrastructure hinders the development of such applications. NFV has the potential to break this vicious cycle and become a crucial enabler for Edge Computing. Standardized server hardware is already being deployed and virtualized by network operators. Therefore, the foundation to run Edge Computing already is there, albeit for a different reason. In addition, should other stakeholders, e.g., cloud infrastructure providers, make increased efforts to enter the Edge Computing market, this competition is likely to accelerate the adoption of Edge Computing in the NFV environment.

CHAPTER 4

Classification and Analysis of Applications

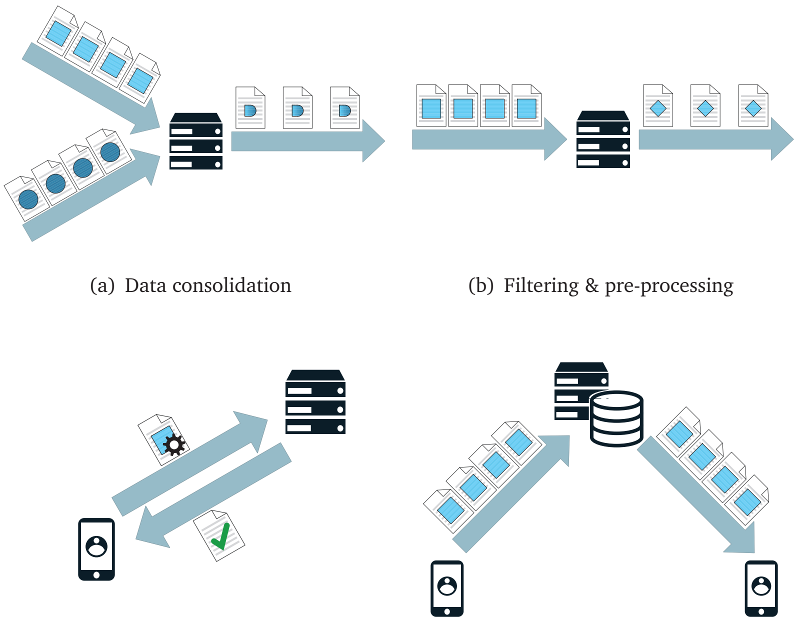
Chapter Outline

This chapter reviews and analyzes the application landscape that revolves around Edge Computing. First, in Section 4.1, we present a systematic methodology for the analysis of applications. More specifically, we introduce four kinds of *components* that applications are comprised of and suggest to classify applications according to the notion of *augmentation*. As a second contribution, using the introduced methodology and the characteristics of Edge Computing (see Chapter 3), we take a detailed look at possible applications for Edge Computing. Section 4.2 analyzes representative use cases for each category of applications and discusses the mapping between an individual application's characteristics and how well they can be served by executing the application or parts thereof at the edge. This helps to get a clearer picture of how the challenges differ for different application types and where using Edge Computing can truly be beneficial. From the results of this survey, we summarize key observations in Section 4.3 and conclude with requirements that will be addressed in the remainder of this thesis in Section 4.4.

4.1 Methodology

4.1.1 Components of Edge Applications

Applications that make use of Edge Computing do so by offloading data or computations to resources at the edge. In addition to this general distinction, we can furthermore take a data-centric perspective and investigate *how* data is transformed and processed. Based on these two aspects, we define the following four components that applications can use: (i) data consolidation, (ii) filtering & pre-processing, (iii) data storage & retrieval, and (iv) computation offloading. They are visualized in Figure 4.1. While the first two concern the flow of data and describe *how* data is transformed, the latter two indicate *what* happens with the data. Those components allow to cover a wide range of applications for which Edge Computing is relevant, e.g., for overcoming devices' limitations (through offloading), enabling collaboration (through sharing of data), or for coping with high-volume data streams (through consolidation and filtering). Besides applications being distributed, the components are also subject to distribution themselves, which entails further challenges, e.g., where to place them. Concrete implementations of the components may have different levels of complexity, and applications can combine multiple of those components.



(a) Data consolidation

(b) Filtering & pre-processing

(c) Computation offloading

(d) Data storage & retrieval

FIGURE 4.1: APPLICATION COMPONENTS[†]

4.1.1.a Data consolidation

Data consolidation combines data from multiple sources and reduces it to a smaller, often more meaningful joint representation. As an example, consolidating data in the context of complex event processing (CEP) [CM12] has the potential to save bandwidth if consolidation operations are placed at the network edge before the streams of data are transferred further [SS13; Gov+14]. One example of data consolidation is averaging sensor data, e.g., providing an average temperature reading over a certain time.

Executing data consolidation tasks at the edge instead of in the cloud has the potential to greatly reduce end-to-end latency and required bandwidth. The amount of bandwidth savings depends not only on the relation between the input bandwidth and the output bandwidth of the data consolidation task but also on the task's location in the network. For example, averaging sensor data on-site might require orders of magnitude less overall bandwidth than averaging the data in the cloud. Furthermore, tight latency requirements might make consolidation in the cloud infeasible in some cases.

4.1.1.b Filtering & pre-processing

The purpose of this component is twofold: discarding irrelevant data (data filtering) and transforming the data or its representation (pre-processing). Since not all data is equally important, bandwidth savings can be achieved by discarding irrelevant data before it is transmitted for further processing. A simple example is thresholding of temperature readings in an application where an alarm should be raised when a certain value is exceeded. In such an application, temperature readings are irrelevant as long as they are within the normal range and thus need not be transmitted. Besides saving bandwidth, reducing data locally can also help to save energy and reduce local storage needs [Gau+13]. In pre-processing, data is transformed from one representation to another. Besides discarding data, which could be interpreted as a special case of such a transformation, other examples are the aggregation of data streams over time, data compression, data alteration, or bridging between formats. For instance, real-time video analysis, a likely "killer app" for Edge Computing [Ana+17; LQB18], has the potential to save vast amounts of bandwidth by only forwarding results of the analysis, e.g., the number of objects in the frame, instead of entire video streams. To give a practical example, a video stream may be encoded to a lower bitrate or faces in the video stream could be blurred for privacy reasons. Powers et al. [Pow+15] demonstrate the use of cloudlets to pre-process data for a face recognition application. Both of these aspects can save bandwidth, depending on the ratio of data discarded and how much data is reduced by the pre-processing. Furthermore, in the case of time-critical data stream processing applications, distributing such operations entirely at the edge can reduce end-to-end latencies substantially [Nar+19; Car+15].

4.1.1.c Computation offloading

Computation offloading is the process of executing a task outside the client device on a remote resource. This task is often resource-intensive. The client transfers the input data and in some cases the code and execution environment and retrieves the result. For example, offloading computation-intensive neural networks [Li+16a; Hu+17; LZC18] or rendering operations [Shi+19] from a mobile device could decrease the execution time and save battery energy. We refer the reader to Section 3.5.1 for a more detailed description of this concept.

4.1.1.d Data storage & retrieval

Applications might want to store data outside of the device for several reasons. First, additional external storage helps to overcome the device's storage limitations. Second, data often needs to be shared across different users and applications. Whenever data is only of local relevance, leveraging the edge, i.e., storing the data on close-by devices, is beneficial for access latencies and bandwidth utilization in the network. Often, this data will be contextual data and short-lived information captured by the user. Storage can either be ephemeral, i.e., short-lived, or permanent. In the latter case and if we consider unreliable devices, replication is required. If data is replicated, the client needs to make a decision from where to retrieve it, ideally considering the abovementioned metrics of latency and bandwidth. Section 4.2.1.c presents several examples of approaches related to data storage and retrieval.

4.1.2 Classification Scheme

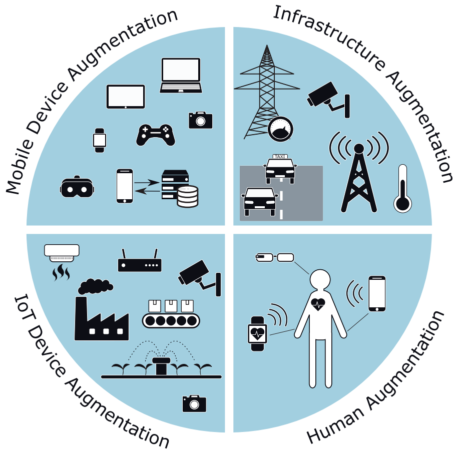


FIGURE 4.2: CATEGORIES OF APPLICATIONS†

To categorize the vast amount of proposed applications for Edge Computing, we classify them into four categories. Contrary to previous works whose classifications are rather enumerative, our classification starts from the very purpose of Edge Computing, i.e., to bring data and computations closer to where data is generated and results of the computations are needed. To do so, we define four categories around the notion of *augmentation*, as depicted in Figure 4.2. We use the term *augmentation* to denote the intertwining of the physical world with the digital world. At the top level, a common categorization of the physical world divides it into animate and inanimate objects. Looking at their interaction with the digital world, the ability to actively and consciously influence the latter is largely restricted to a subset of the animate world, i.e., humans. Therefore, our classification starts from the distinction between *humans* and *things*. For humans, we further distinguish between separate (mobile) devices used and/or carried by the user (*mobile device augmentation*, Section 4.2.1) and devices that are interwoven to a higher degree with the user, e.g., devices that are worn or connected and affect the user's body function (*human augmentation*, Section 4.2.4). For *things*, following the widespread IoT terminology, Section 4.2.3 defines one category as *IoT device augmentation*. However, we also want to stress the benefits of Edge Computing in enhancing the smartness of public spaces and larger environments as opposed to single devices in closed ownership. Therefore, we also define the category of *infrastructure augmentation* (Section 4.2.2). We acknowledge that some use cases may be considered to belong to more than one category; however, we share this issue with the vast majority of categorizations in every domain. Nevertheless, we argue that our classification has the advantage of being simple, inclusive, and at the same time open to fit future applications.

4.2 Application Survey

For each of the categories defined in Section 4.1.2, we describe representative use cases and analyze the benefits of Edge Computing with regards to the four advantages as defined in Section 3.1. Note that regarding energy-saving and energy efficiency, we only consider the potential energy saving and hence battery life prolongation of end devices as this is often more relevant than the total energy footprint. Energy savings in network equipment are sufficiently represented by the bandwidth criterion.

We do not restrict our analysis to papers where Edge Computing has been proposed or that are currently associated with Edge Computing use cases but include others for which at least some benefits of Edge Computing apply. Furthermore, following our taxonomy discussed in Section 2.1, we also include literature that relates to Fog Computing. This helps to get a comprehensive overview of potential use cases for Edge Computing. At the end of this section, we summarize our findings for the most pertinent use cases. Those references that appear in Table 4.1 are marked with an asterisk (*) throughout the following subsections.

4.2.1 Mobile Device Augmentation

The applications outlined in this section are specifically targeted at the consumer's mobile devices, such as smartphones, tablets, or head-mounted devices. Specifically, we look into mobile gaming (Section 4.2.1.a) and emerging virtual/augmented reality applications (Section 4.2.1.b). In addition, new applications and usage patterns require appropriate strategies for data storage and caching (Section 4.2.1.c).

4.2.1.a Mobile gaming

In gaming, we observe the trend towards (i) more mobile games (i.e., games played on a connected handheld device), (ii) more games that interact with the player's environment or other players in proximity, and (iii) business and deployment models based on *Gaming as a Service* (GaaS). As described in [CCL14], GaaS is the concept of providing scalability and overcoming hardware limitations through modularization of the game. For example, certain functionality (e.g., rendering) is migrated from the mobile device and offloaded to a server. Functionalities like rendering are computationally intensive, and hence, offloading them can significantly improve battery life.

Most games are highly interactive and, thus, players expect crisp responsiveness. The responsiveness is influenced by the computation and communication time of the offloaded components and—in some cases—by the latency to other players. Pantel and Wolf [PW02] claim that depending on the type of game, only 50 ms–100 ms of delay is tolerable. As analyzed by Choy et al. [Cho+12], the cloud will be unable to meet the latency requirements of gaming applications (assuming a target latency of 80 ms). To solve this issue, the authors propose to extend the cloud infrastructure with local content delivery servers to serve users' demands. This has led big companies to extend their data centers towards the edge. Plumb and Stutsman [PS18]* demonstrate how exploiting Google's edge network can reduce the latency for massively multiplayer online games. Specifically, they define an *area of interest latency* as the latency between players that interact in the virtual world. The authors report an improved mean latency from 52 ms to 39 ms and for the 99th percentile an improvement from 110 ms to 91 ms. As shown before, this can make the difference between an enjoyable experience and an unplayable game.

For these reasons, extending the infrastructure for mobile gaming to the edge makes sense. In addition, tasks like rendering typically require only small data as input (e.g., the user's position, field of view or current action) while the size of the returned data (in this example the rendered object) is much larger. This observation matches the asymmetric down- and uplink bandwidth that end users today have in cellular networks. Messaoudi et al. [MKB18b]* present a general framework for offloading parts of a modularized game engine. Lin and Shen [LS17]* extend cloud gaming with fog nodes that are responsible for rendering game videos and streaming them to nearby players. Similarly, the work of Kämäräinen et al. [Käm+14] supports the deployment of game services in hybrid (public, enterprise, private) cloud infrastructures. Using local processing, they were able to almost halve the delay. Furthermore, their results indicate that connecting to a cloud via WiFi is less detrimental to the device's energy consumption compared to 4G cellular networks.

Cai et al. [Cai+18]* investigate a scenario in which neighboring players cooperate in a game. They advocate cloudlets in the vicinity of players to reduce bandwidth and latency bottlenecks, and also consider the energy dimension, both for the mobile device and the overall energy cost for data transmission. In many games, players are mobile, e.g., when the game requires them to interact with their environment or visit different locations. Hence, the issue of migrating offloaded parts of the game arises. Braun et al. [Bra+17b] propose an application-level migration technique for latency-sensitive gaming applications. The server is transparently live-migrated during gameplay. The authors argue that the advent of 5G networking will provide ultra-low latency and high-bandwidth to mobile devices. Hence, it would make sense to locate edge game servers at those locations.

4.2.1.b Augmented reality and virtual reality

Augmented reality (AR) and virtual reality (VR) have gained enormous traction due to the advent of new consumer devices such as the Microsoft *HoloLens* or HTC *Vive*. In augmented reality, virtual objects are integrated into the environment [Azu97]. Augmented reality applications extend the user's real-world view by inserting virtual objects into the environment, related to one's context and movement. Virtual reality (VR) is defined by creating a sense of *presence* in simulated environments [Ste92]. Both variants are typically interfaced to the user via a head-mounted display (HMD). Applications leveraging VR and AR mainly fall into the categories of information (e.g., in retail [Cho+16] or for tourist guidance [Tal+17]), education [Wu+13], and entertainment [Tho12]. Common to all of them is that computationally heavy tasks need to be carried out in order to render and understand a scene.

AR/VR applications are subject to stringent real-time demands on their responsiveness. For head-tracked VR, it has been shown that the JND[1] for latency discrimination is 15 ms [Man+04]. Especially in VR, motion sickness can occur if the delay between tracking the movement and rendering the scene exceeds this value.

Because high-framerate 3D rendering is computationally demanding, many headsets carry out those tasks on a standalone computer, connected to the headset through a cable. Removing this cable is desirable for the user experience, but challenging due to the high data rates a wireless channel must guarantee. Some attempts have been made by using new wireless communication technologies such as mmWave [Aba+] or by partitioning the computations between the HMD and the rendering server [CCK18]. The latter is required because computational capacity (especially GPU) remains limited on end devices. At the same time, the stringent latency requirements prohibit offloading to the cloud. Hence, the challenge is to design good offloading strategies to nearby rendering servers while coping with current wireless communication technologies. Braud et al. [Bra+17a] have analyzed these challenges with a focus on mobile users. Furthermore, as outlined in [Tri+19], battery life and latency form a tradeoff.

Despite these challenges, a few solutions to leverage Edge Computing in AR and VR have been proposed. Elbamby et al. [Elb+18] have envisioned a combination of Edge Computing for computation and mmWave for communication as a crucial enabler for wireless VR. Lai et al. [Lai+17]* have presented the idea of a cooperative renderer. The authors base their system on the observation that less-predictable scene updates are typically more lightweight, and therefore, those are rendered at the edge. For highly predictable scene changes, pre-rendered frames are fetched from nearby locations. In addition, compression and panoramic frames are used to reduce the load on the wireless link. Similarly, Shi et al. [Shi+19]* have presented a rendering scheme that saves more than 80 % of bandwidth and therefore enables the delivery of VR content over 4G wireless links. Liu et al. [LH18] have presented *DARE*, a novel network protocol that enables mobile users to dynamically change their AR configuration. Specifically, it adapts to changes in network conditions and load on edge nodes—both crucial in dynamic edge environments. Tirelli et al. [Tri+19] have used an approach based on NFV. They have developed a framework for live video augmentation by extracting and injecting video streams from or into network flows.

Besides partitioning workload only between the end device and one edge node, some workloads benefit from distributed processing across several cloudlets. For example, Bohez et al. [Boh+15]* reconstruct 3D maps from the depth cameras of AR headsets. They do this by partitioning sub-models across geographically distributed cloudlets.

Gaming with VR/AR is a popular application scenario, and due to its specific characteristics, we review existing work independent from Section 4.2.1.a. Viitanen et al. [Vii+18] have presented a rendering scheme for real-time VR gaming that saves energy and computational load on the end device. The rendered views are encoded as HEVC[2] frames and transmitted based on the user's field of vision. In [Zha+17], the authors have explored the scalability issue of massively multiplayer games and present a hybrid approach in which changes in the local view of a player are processed at the edge, while global game updates are performed in the cloud. In addition, colocating multiple players that share a similar view on the same edge

[1] just noticeable difference

[2] High Efficiency Video Coding

node increases the efficiency of the proposed system. Zao et al. [Zao+14] have developed an AR game with a brain-computer-interface that processes EEG[3] brain activity in real-time. Classifying those readings into game actions is a computationally intensive task. The authors use a combination of Edge Computing and Cloud Computing for this. While the classification is done at the edge, the underlying models in the cloud are continuously adapted according to the sensor readings.

In summary, AR and VR applications are two of the most relevant use cases for Edge Computing, as they combine four important characteristics that benefit from edge deployments: strict constraints on the latency, high-bandwidth data, computationally intensive tasks, and battery-powered end devices.

4.2.1.c Data storage, content delivery, and caching

Virtually all connected applications need to store or retrieve data from outside the client device. At the beginning of this section, this was defined as one of the components that applications use. We now describe in detail the possible types of storage and caching services that can be offered at the edge.

Content Delivery Networks (CDN) [Dil+02] distribute content caches across data centers in different geographic regions to provide highly available and performant content retrieval for users. Ericsson forecasted back in 2013 that in 2019, 50 % of mobile traffic would be video traffic [Eri13]. Hence, content delivery networks today to a large extent serve video traffic [MJ16]. This demand for video not only means users have high expectations for the service quality, but also that large-volume data is a huge burden for infrastructure providers and hence, an important incentive to place caches within the access network [Bec+14]. Ahlehagh and Dey [AD14]* have proposed both reactive and proactive caching strategies for video content at the radio access network. Bastug et al. [BBD14] have investigated the role of proactive caching in 5G networks. Another example is the retrieval of websites [Zhu+13]*. Zeydan et al. have [Zey+16] proposed proactive content caching based on content popularity in 5G networks. Approaches like [App+10] assume a global knowledge of the content popularity for each location. Because edge environments typically feature user mobility, these approaches have limitations with regards to their scalability. A solution to this might be collaborative approaches as presented in [LYS16]*, where base stations collaborate in replicating content to improve the overall cache hit ratio. Tran et al. [Tra+17]* have presented a strategy for collaborative caching and processing of on-demand video streams.

Up until today, little research has linked content storage and caching to the domain of Edge Computing. Drolia et al. [Dro+17] have presented *Cachier*, a caching system for image recognition applications. It balances the load between the edge and the cloud and exploits the spatio-temporal properties of requests. Psaras et al. [Psa+18]* have advocated the placement of local storage on WiFi access points to buffer IoT data prior to cloud synchronization. Lujic et al. [LMB17]* have proposed a storage management framework for edge analytics. The goal of the framework is to balance the quantity of stored data and the resulting quality of the data analytics, given limited storage capacities at the edge. Such caches can also be used for the distribution of mobile apps and app updates [Bha+15b]*,[Bha+15a]*.

As end devices generate more data, the path of data dissemination changes to a *"Reverse CDN"* [Sch+17]*, [MSM17]*, i.e., the decision is where to store data gener-

[3] Electroencephalography

ated by those end devices in the network. This decision depends on where the data is to be used and shared. Therefore, contextual information, such as the user's location is useful to make this decision. For example, we can imagine sharing content from large-scale events or tourist sites. In Chapter 9, we will present a framework that uses contextual information of the user to make storage decisions. Simoens et al. [Sim+13] have presented *GigaSight*, a framework for the decentralized collection of videos through cloudlets. Before the videos are indexed and made available, cloudlets remove privacy-sensitive information from the video. The authors have conducted experiments, measuring the throughput and energy consumption and, based on the results, modeled a tradeoff for the allocation of resources between the cloudlets performing the different computation steps.

Compared to low-latency processing tasks, the benefits of Edge Computing for storage and caching are less striking and require careful decision-making. Furthermore, using edge nodes for storage can raise the question of resilience in view of their inferior reliability [MRS19] compared to large data centers. However, in cases where data has to be transferred away from end devices, e.g., because it is to be shared, and if that data is only relevant in a certain (geographic) area, Edge Computing can save bandwidth if the data is kept in proximity. As an example, Hao et al. have presented *EdgeCourier* [Hao+17]*, a framework for live document synchronization. The authors have demonstrated the reduction in network bandwidth usage by performing incremental synchronization at edge nodes. Another example is the work of Hu et al. on face recognition [Hu+17]. They have observed that only a subset of biometric features is relevant to identify a face. The paper suggests extracting those features at the cloud and retaining only those at the edge necessary to perform the recognition.

As soon as personal user data is involved, privacy and data security become relevant. The *Databox* project [Cha+15]*, [Mor+16]* aims at providing a personal data management framework. It enables individuals to manage their personal data and make that data available to others, while retaining control over its usage. Most importantly and in contrast to today's approaches, data is not handed over to third parties. Instead, access and processing is mediated through the Databox. The authors envision a distributed system of personal Databoxes, hosted on a variety of devices, e.g., wireless gateways, lamp posts, and cars. The Databox consists of several components that are logically separated and can run on different physical devices. Access to different sources of data is also logically separated to provide additional security. The project is a good example of how edge infrastructure can help to preserve data security and privacy. However, as noted in [Per+17b]*, several challenges remain, such as capturing one's privacy preferences, implementing shared ownership of data, or how to approach risk-benefit negotiations for sharing data.

4.2.2 Infrastructure Augmentation

In the following, we refer to infrastructure as basic services and facilities that we build our lives upon. This basic infrastructure encompasses everything from our electrical grid (see Section 4.2.2.a) up to emergency services (see Section 4.2.2.d). This infrastructure is especially important and challenging in urban areas as the trend of growing urbanization brings new challenges, such as traffic, pollution, and safety [LR17]. The term *smart cities* [Sch+11] has been used to describe concepts that use information technology to augment infrastructure in urban areas. Schleicher et al. [Sch+16b] define a smart city as a reactive system that makes decisions based on massive amounts of data. Smart cities connect people and objects in order to create services that enhance the quality of life for citizens of urban environments [Zhe+14a]. Furthermore, monitoring and large-scale data analysis can provide valuable information for municipalities and policymakers, e.g., to plan traffic and transportation (see Section 4.2.2.b). The raw data is collected from sensors that are deployed in the environment. Besides static deployments, humans can also be incentivized to serve as data providers [Sch+12], e.g., by providing sensor data through their phones [CFB14]. One application domain where this approach has proven useful is in monitoring the environment (see Section 4.2.2.c).

Previous works in the domain of smart cities have proposed to integrate the captured information through Cloud Computing [Heo+14; PLM17]. However, they have not considered the potential benefits of using nearby cloudlets, especially for analytics on high-volume data. A prominent example are video streams from cameras that are ubiquitously present in today's cities. While tolerable delays for smart city applications vary greatly [Zan+14], the mere number of sensors will prevent cloud-based solutions from scaling. Perera et al. [Per+17a] have further surveyed different types of smart city applications that benefit from edge deployments, with a focus on the communication between devices.

4.2.2.a Smart grids

Energy grids are currently in a state of transformation towards so-called *smart grids*, driven by ecological, economical and political goals. Two important characteristics of a smart grid are (i) information and communication technology is embedded into the energy grid, (ii) its decentralized nature (in contrast to today's strictly hierarchical organization), and (iii) a shift towards *prosumers* that both consume and produce energy. A smart grid is envisioned to support a distributively organized control structure instead of being one large energy grid with central control across all tiers. In particular, this structure is envisioned to consist of multiple cells, which can be controlled individually [How+17].

As a way to realize the next evolutionary step of cellular energy grids, a *holonic* approach has been suggested [NB12]. Holons represent entities in a system that are simultaneously a part and a whole. Consequently, a hierarchically organized system structure (called *Holarchy*) emerges, where on different levels, holons exist that encompass other holons, while being part of higher-level holons themselves.

Implementing a Holarchy is challenging, given that it requires components for monitoring and automated control [Fre+13]*. To establish such a concept, Edge Computing is a suitable paradigm as it supports data aggregation and filtering, which are both essential to realize the concept of holons. For instance, local consumers can optimize the control of the appliances in their smart homes to reduce their electricity bills [AMM15]. Simultaneously, this goal may conflict with the process of balancing demand and supply in the overall grid. These problems can now be addressed either locally, using the local controllers responsible for managing individual holons, or the higher level holons can address the issue as they have a larger view on the network.

The decision-making in smart grids must solve complex optimization problems, like optimally controlling distributed energy sources. One way to tackle the complexity of these problems is to reduce the problem size and, consequently, speed up the optimization process. Edge Computing devices can be used as local controllers for holons, which are capable of executing optimization algorithms. Examples of suitable algorithms are presented in [PSM10]*. Moreover, such optimization tasks need to be performed quickly, as demand and supply deviations in the grid can contribute to the destabilization of the whole energy grid. Edge Computing devices facilitate low-latency communication and hence, using Edge Computing, these problems can quickly be addressed locally, without the need to send the necessary information to a cloud service or a central control center.

Although not obvious at first glance, smart grids contain sensitive data with regard to one's privacy. For example, the authors of [Rei+12] have shown that the readings from smart meters can reveal the individual home appliances by analyzing the aggregated smart meter readings. Edge Computing devices in combination with smart energy storage technology can address this problem by scheduling the charging of the storage and the consumption of the appliances in a way such that the devices only use the electricity that is currently stored in the home. Consequently, the power consumption profile of the house itself appears to the outside only as the charging behavior of the local energy storage device.

4.2.2.b Smart transportation & connected cars

In big cities, optimizing traffic conditions is a crucial task as it results in a tremendous impact on one's quality of life. Like in other types of smart city applications, many primary sensors for traffic-related applications are cameras, whose video streams require further processing. For example, we can think of a smart traffic light that adapts its signal cycles to the actual traffic conditions in certain lanes of the intersection [Gha+16]*. While energy is not an issue in these scenarios (since sensors are fixed deployments and connected to a permanent power source), this use case requires complex tasks like object recognition (to identify cars) and tracking that might not be feasible given weak built-in hardware. Analysis of live feeds can also be used to detect traffic anomalies [Ana+17] or to recognize license plates [Yi+17]. The traffic light example can also be linked to emergency response use cases, e.g., to help making way for an ambulance by setting traffic lights to free up the route and warn others about the approaching vehicle [Nun+15]*. In the latter use case, time criticality clearly becomes more important. This is also the case for the detection of immediate road hazards, e.g., as shown in [CDO19]*.

Long-term analysis of urban data can be used to detect and improve flawed urban planning or to identify areas where dangerous situations regularly occur [Zhe+11]*,[Sat+17]*. In these use cases, the long-term analysis would most likely be carried out in the cloud due to the larger amount of available resources. However, we argue that data collection on such a big scale would not be possible without Edge Computing, due to necessary pre-processing steps (e.g., encoding a video to a lower bitrate or aggregating measurements). Qi et al. [QKB17]* have introduced an Edge Computing platform using on-board computers on public transport vehicles. The platform collects data (e.g., by intercepting WiFi probe packets from phones) to gain insights that help public transport operators to devise better plans (e.g., by identifying popular stations and assessing vehicle occupancy). Such information can be sensitive as it allows to identify and track individuals. Hence, in this case, the edge infrastructure could also be in charge of performing the anonymization of sensitive data. Besides static planning, this high-volume data could also be used for real-time updates, e.g., to provide passengers with predicted arrival times. This data could be distributed to edge nodes at the relevant location (e.g., to the corresponding WiFi hotspots at a stop). The location-awareness of edge nodes can also be used for applications like toll collection [Aba+15] or finding parking spaces [Vil+15b; Awa+19].

Besides improvements in the planning of city traffic, we see a trend towards vehicles themselves being equipped with more sensing and computing capabilities. In many cases, these components are not isolated but transform the vehicle into a *connected car* that can interact with its environment and other vehicles. Such cars can form vehicular ad hoc networks (VANET) [HL08] that communicate with each other or through some close-by infrastructure—often termed roadside unit (RSU). Besides RSU, UAVs have also been proposed as a means to relay communications between cars and/or infrastructure [Men+17]. Datta et al. [DBH15]* have presented an infrastructure with a Fog Computing layer, located at RSUs and M2M-gateways. The type of data and its importance varies in such networks [Sch+08], from simple information services (e.g., information about current traffic conditions) to critical, safety-related events (e.g., warnings about emergency situations or sudden breaking of cars ahead). An example of the latter is the work of Cozzolino et al. [CDO19]* that uses an edge infrastructure for black ice road fingerprinting. Liu et al. [Liu+18] have demonstrated that complex tasks, such as recognizing attacks in ridesharing services can be done with little energy impact on the end device. Edge Computing can also be an important enabler for autonomous driving, e.g., by disseminating data from RSUs to vehicles [Yua+18]*, or by processing information like point clouds captured by LIDAR sensors [Qiu+18]*. Besides such latency-critical and compute-intensive tasks, edge capabilities can also be used for early data aggregation to save core network bandwidth. In this domain, Lochert et al. [Loc+08] have presented an aggregation scheme for VANET traffic information. As outlined in [BOE17], strong security mechanisms need to be in place if edge nodes are involved in the control of vehicles in order to make such systems resilient against attacks. Raya and Hubaux [RH05] have provided a detailed analysis of threats and security architectures in VANETs.

4.2.2.c Environmental monitoring & waste management

Pollution is one of the main problems in growing urban areas, with drastic impacts on people's health. As of today, many kinds of pollution are only measured at very few locations and/or estimated via models based on historical data [ZLH13]*, [Zhe+14b]*. Hence, they often fail to provide an accurate and actionable view on current situations. However, accurate, real-time information is crucial, both for policymakers and citizens to make informed decisions (e.g., for patients with respiratory diseases or for decisions to restrict traffic). Dutta et al. [Dut+17]* present *AirSense*, a crowdsensing-based air quality monitoring system. Especially for such opportunistic sensing campaigns, where the location of the data is unknown a priori and constantly changing, the deployment of Edge Computing resources is both beneficial (because the data is typically high-volume and can be aggregated locally) and challenging (because of the geographic dispersion and the users' mobility).

While environmental data, in general, has a rather low rate of data generation [SBH16], scalability remains an issue [MAŽ18]*. Edge Computing can ensure the scalability of a large-scale sensing system by processing the data close-by and keeping the results within the sensing area. Aggregated or coarse-grained data (e.g., by reducing the temporal resolution) can be sent to the cloud, while the raw sensor readings are processed at the edge. Aazam et al. [AH14] have advocated data trimming to reduce unnecessary transfers to the cloud by using a smart gateway that sensing nodes are connected to. Edge infrastructures have also been used to opportunistically deploy different sensing tasks [Tsa+17]. Zheng et al. [Zha+18b] have presented an allocation scheme for sensing tasks on Edge Computing nodes. The authors outline the benefits of an edge deployment w.r.t scalability, bandwidth requirements, and better utilization of edge nodes' computing power.

Similar to air pollution, noise pollution is a big problem with adverse effects on people's health. Maisonneuve et al. [Mai+09]* and Schweizer et al. [Sch+12]* have proposed deployments in which citizens measure noise through their mobile phones and upload the measurements to a cloud-based service for access and analytics. Besides the potential benefits of edge deployments w.r.t. latency and bandwidth, we also need to consider the privacy aspect. Whenever data is collected by volunteers, privacy has to be guaranteed, otherwise people might not be willing to participate in sensing campaigns. Measurements must inherently contain the users' locations, and hence, this would allow tracking user locations. By processing such data at the edge, data can be anonymized early or, alternatively, noise can be introduced into the raw data, such that it does not impact the results, but cannot be linked back to an individual. Here, the distributed nature of an Edge Computing infrastructure itself can be exploited. Marjanović et al. [MAŽ18] have demonstrated how partitioning data and distributing its processing can help in reducing privacy threats.

Waste management is a complex process in today's cities and includes the collection, transportation, processing, and disposal of waste. Optimizing these processes can save a city tremendous amounts of money and resources. One way is to optimize the transport routes of garbage collection trucks. Normally, these operate at fixed schedules, as no real-time information about filling levels of waste containers is available [Nuo+06]. Sensors mounted on trash containers could report their filling levels and infer if they need emptying. Based on aggregated data from a neighborhood, the garbage trucks' routes can be optimized. Furthermore, municipalities can provision different sizes of garbage trucks in order to optimize the collection process [AZM15]*. Cloud-based solutions have been proposed [Per+14b]*, [Aaz+16]*, [Med+15]*, however, aggregating the sensor readings at the edge would save bandwidth. This becomes more important if data is annotated with photos or voice messages, as suggested in [Med+15]. Latency and privacy, however, are less of a concern in this use case.

4.2.2.d Emergency response & public safety

Detecting emergency situations can be done by inferring events from sensor sources. As soon as an emergency is detected, first responders need to be alerted and directed to the scene. To do so, platforms provide situational awareness and connect first responders to the required data sources. Chung et al. [Chu+13]* have presented a cloud-based platform, while Aazam and Huh [AH15]* have extended this idea to incorporate an intermediate fog layer that is capable of pre-processing the data and overcoming delay problems. Depending on the type of incident, the appropriate emergency departments are notified. Mobile base stations can furthermore serve to notify citizens about a threat. For example, Sapienza et al. [Sap+16] outline a use case where a fire is detected based on sensor readings and video analysis, and this information is forwarded to car navigation systems in order to alert people.

A distributed infrastructure of edge devices can furthermore be leveraged as an emergency infrastructure in case of disasters. In case of a breakdown of the communication infrastructure, cloudlets hosted on private devices like routers can act as emergency devices, providing both computation and communication capabilities [Meu+17c]*. Satyanarayanan et al. [Sat+13]* outline a potential use case for such emergency cloudlets in disaster recovery, in which responders take photographs that need to be stitched together in order to obtain a complete overview of the area.

Efforts to increase public safety (e.g., by either preventing or quickly detecting crimes) mostly rely on surveillance. In many cases, the raw data consists of video streams, which are then analyzed. Canzian et al. [CS15] have presented a hierarchical classifier system for surveillance applications. As the authors point out, the characteristics of the tasks—distributed sources and tasks as well as a high computational complexity—make it necessary to leverage distributed and heterogeneous processing nodes. This definition perfectly reflects the Edge Computing landscape. Chen et al. [Che+17a]* have presented a system for real-time surveillance and tracking of vehicles to detect speeding using a drone. Similarly, Xu et al. [XGR18] use a geo-distributed Fog Computing infrastructure for vehicle tracking. Mihale-Wilson et al. [MFH19] have investigated the protective effect of street lamps if they are augmented with functionalities such as video surveillance and gunshot detection via microphones. Their results suggest an increase in safety in areas where such lamps were installed. Just-in-time indexing of video streams across several cloudlets has been demonstrated in [Sat+17]*. A practical use case for just-in-time video indexing could be the search for a missing person.

Because all these use cases involve multiple streams of high-volume data and heavy computations, edge deployments are beneficial in terms of latency and bandwidth. Besides offloading computations, data pre-processing is also relevant to some security-related applications. For example, Stojmenovic [Sto12]* has proposed to partition tasks for biometric identification between the mobile device and the cloud.

In all those use cases—especially when video streams are involved—the citizen's privacy is exposed, and personal information is analyzed. Contrary to other use cases, the privacy-critical information is not incidentally contained in the raw data, but it is the reason for capturing it in the first place. Hence, the positive impact of Edge Computing in this application domain is limited. At the very best, we could envision enforcing policies to delete personal information once it has been processed by a trusted edge node.

4.2.3 IoT Device Augmentation

The IoT refers to connected objects that are able to interact with each other and hence, extend the Internet to the physical world [AIM10]. Originally closely tied to RFID[4] technology [Wan06], today the IoT encompasses all kinds of sensors, machines, and appliances. An extensive survey about the IoT and its enabling technologies can be found in [AlF+15]. The data volume and latency requirements of future IoT devices will likely be challenging to transfer and process at central clouds [PM17; San+14]. In this section, we focus on IoT deployments in three settings: homes and buildings (Section 4.2.3.a), industrial applications (Section 4.2.3.b), and agriculture and farming (Section 4.2.3.c).

[4] Radio Frequency Identification

4.2.3.a Smart home & smart building

The terms *smart home* and *smart building* describe the concept of collecting data within a building and using it to automate and optimize various aspects of the building. The IoT offers great potential to lower energy/water consumption and increase security and comfort through coordinated management of HVAC[5] systems, lighting, electrical outlets, and various connected devices [Cas14]. Examples for such devices are smart locks, surveillance cameras, TVs, household appliances, or environmental sensors. These devices and building systems produce large amounts of sensitive personal data streams [Shi+16]. One example of a smart home task is the aggregation and pre-processing of home video surveillance streams [San+14]*. Such a case was examined by Abdallah et al. [AXS17]* in a prototype system. Their results indicate that storing all sensor data in the cloud has a negative impact on the available bandwidth in the home, pointing to Edge Computing as a solution.

Edge Computing could also be used for the aggregation of IoT data, and thus, enable cooperation between devices by using sensors and physical capabilities of multiple devices to complete a task [Yi+15; Pan+16]. Such a task could be sending a robot vacuum cleaner with a camera to check on suspicious motion through video analysis. Vallati et al. [Val+16] have envisioned an MEC architecture for smart homes that builds on LTE with device-to-device communication for data locality and low latency. Storing and processing smart home data in-home could also resolve the issue of transferring privacy-critical data [Per+17b; Mor+16] and opens up the possibility to combine data from IoT devices with personal data from other services to provide higher-order, yet privacy-aware services. As proposed in [Fer+18]*, Edge Computing and Fog Computing could also provide the missing link between various building subsystems that are usually implemented independently. Thus, the resulting integration and interoperability between the individual subsystems, combined with data analysis at the edge, could enable new smart services, like the activation of smart devices when solar power is available.

4.2.3.b Industrial Internet of Things

One goal of the *Industrial Internet of Things* (IIoT)[6] is to improve quality control in manufacturing and increase productivity [Wan+18a]*, [LGS17]*. Collecting and analyzing vast amounts of data from production processes can be a tool to find inefficiencies and optimize production processes [LGS17]*,[AZH18]*. However, since it is often impractical to store and analyze all collected data at the cloud in real-time, Fu et al. [Fu+18]* suggest to pre-process, aggregate, and store time-sensitive data on edge nodes, while storing less time-sensitive data in the cloud for later analysis. Early detection of mechanical problems with production machines has the potential to prevent both machine failures and production quality issues [Oye17]*.

Monitoring machines produces large amounts of real-time data that lends itself to being analyzed at the edge to monitor machine health or to predict tool wear and service intervals in real-time [Oye17]*, [Wu+17a]*. Another goal of the IIoT is to enable the production of highly customizable products on dynamically scheduled production lines, which can also profit from the low-latency property of Edge Computing [Wan+18a]*. Lin et al. [LY18]* have presented a further application

[5] Heating, Ventilation, and Air Conditioning

[6] We use IIoT synonymously with industry 4.0 / advanced manufacturing

of Edge Computing for IIoT, namely the real-time scheduling of logistics in highly automated warehouses.

4.2.3.c Agriculture & farming

Smart farming (or precision farming) targets the management of crop and livestock. Many applications in this domain revolve around automation, e.g., for watering crops or feeding livestock. Monitoring of environmental parameters and tracking of animals can furthermore ensure a timely reaction if abnormal events are detected. A delayed reaction can cause damages and production losses [ADH18]* and can impact the entire supply chain. More complex tasks include the use of machine learning methods for yield prediction or disease detection [Lia+18]*.

Pastor et al. [Pas+18] have presented a system for distributed computing in agriculture that includes three layers: things (i.e., the individual sensors and subsystems), edge (responsible for the control of subsystems), and fog (providing local storage and analytics). Tasks carried out at the local edge and fog layer include data filtering, classification, and detection of events. The primary reason for carrying out those tasks at nearby layers is latency, as the system tries to optimize itself in real-time. Another example of a layered system can be found in [Car+17]*, where Raspberry Pi computers are deployed as sensors in the environment and on animals to monitor temperature and movement. This data is processed at an edge layer on the farm itself, whereas cloud infrastructure is used to collect long-term statistics. Besides a strong focus on the communication technologies at the edge layer, the work of Ahmed et al. [ADH18] proposes a hierarchical structure of fog nodes and suggests to carry out computations at local gateways. As one of only a few approaches, it also considers the energy aspect of this local processing. However, their main arguments for edge processing are reduced latency and bandwidth limitations.

Wolfert et al. [Wol+17] see big data analytics as a major disruption for farming and have identified real-time analytics of agricultural data as a key challenge. They emphasize the issue of data quality, i.e., errors in the raw data make operations such as denoising and transformation necessary before further processing. From the perspective of saving upstream bandwidth, edge deployments are beneficial in such use cases. Similarly, Ivanov et al. [IBD15] have observed that many of the sensor data gathered in smart farming contain redundancies that need to be fused before being pushed to a centralized entity.

While a number of edge-enabled systems exist, we observe that most are closed, hybrid deployment, i.e., the intermediate edge layer is deployed on-premise and not part of a public Edge Computing infrastructure.

4.2.4 Human Augmentation

Human augmentation is the process of improving the well-being and capabilities of humans. This augmentation can be done by oneself, e.g., through quantifying, analyzing, and subsequently influencing one's behavior (Section 4.2.4.a), or in the context of healthcare-related applications (Section 4.2.4.b). Moreover, Section 4.2.4.c shows how human cognition can be augmented or assisted.

Privacy and data security are critical concerns in these types of applications due to the intimate nature of the data. As shown by Fereidooni et al. [Fer+17], today's cloud-based services fail to provide data integrity, authenticity, and confidentiality.

However, those factors are critical for the acceptance of such services. Besides the trustworthiness of the nodes that store or process the data, fine-grained access control policies for the remote access and forwarding (e.g., a physician can forward a patient's data to a pharmacy) should also be implemented. To realize this, we can image a network of federated, trusted edge nodes across different organizations.

4.2.4.a Quantified self

With new affordable personal devices, people have gained an interest in collecting and analyzing data about their own bodies and behaviors. This concept is commonly referred to as the *quantified self* [NS14]. Besides getting a deeper understanding of oneself, this data can also be useful in many health-related aspects, e.g., for personalized medicine or preventive medicine [Swa12]. Users are often interested in aggregate values, such as the total walking distance for a single day. Such aggregation can be performed at the edge. If the users wish to rely on cloud services, only those aggregated values are sent to the cloud. Aggregating and storing data is an important use case for personal fitness trackers that count steps, monitor one's heart rate, or analyze sleep patterns. This type of wearable fitness technology is a big part of the quantified self community today [Gil16].

Schmidt et al. [Sch+15]* present a digital fitness coach to support individuals in achieving fitness goals. The system generates training plans and is able to adapt them, e.g., depending on a user's movements. Among other data, data from tracking devices is used. Bajpai et al. [Baj+15]* use heart-rate readings from wearable sensors to track physical activity, map the activity to calorie consumption, and estimate the cardio-respiratory fitness of a person.

4.2.4.b Precision medicine

The umbrella term *E-Health* describes applications that make use of information systems in order to improve people's treatments and overall health. The concept of *connected health* [Shi+16] describes how different actors (e.g., patients, hospitals, and physicians) are linked in order to improve their services. In this section, we summarize the above concepts as *precision medicine*. It has been noted that healthcare is one of the prominent applications for future information technologies [BZ11] and that cloud-based healthcare can help to reduce the overall costs of healthcare [Ala+10]. For a detailed survey of healthcare-related applications, we refer the reader to Kraemer et al. [Kra+17]. Bui et al. [BZ11] have identified three requirements for healthcare applications: (i) interoperability, (ii) reliability and bounded latency, and (iii) privacy, authentication, and integrity. Edge Computing can help with the latter two. As an example, latency is especially critical if the application is tied to a cognitive process or a time-critical control loop (see Section 4.2.4.c).

By monitoring the parameters of a patient and combining information from health sensors with other ambient sensors, health-related issues can be detected. One example is fall detection for stroke patients [Cao+15]*. To perform these tasks, large amounts of raw data have to be analyzed or complex features need to be extracted [Gia+15]. Often the monitoring is part of a sense-process-actuate loop, i.e., whenever an event or anomaly is detected in the monitoring phase, a (timely) action has to be taken. For telesurgery, latencies below 200 ms are optimal [Xu+14]*. Such constraints can be challenging when relying on distant clouds. For other applications such as ECG monitoring, delays in the order of several seconds might be acceptable [AG10]*. For less critical parameters, storing aggregate values for later retrieval is sufficient. Amraoui and Sethom [AS16]* have presented an architecture for patient monitoring in Wireless Body Area Networks (WBAN) that makes use of cloudlets for close-by processing of sensor data. Hybrid edge-cloud systems also exist, e.g., Althebyan et al. [Alt+16] have presented a scalable health monitoring system that uses both Cloud Computing and Edge Computing. Similarly, Azimi et al. [Azi+17] have developed a 3-tier system, in which tasks are partitioned among the tiers. For example, the training procedure for machine learning algorithms is carried out in the cloud, whereas the resulting classifiers are deployed closer to the sensors.

4.2.4.c Cognitive assistance

The idea of cognitive assistance comes from the vision of augmenting human cognition through computing systems [Sat04]. These types of applications are, for example, useful to assist the elderly who suffer from deteriorating senses or for patients with neurological diseases, such as Alzheimer's. Applications that aim to assist or substitute the cognitive tasks of humans should preferably not be slower than humans. This is challenging, given that for instance, humans recognize familiar faces in the order of a few hundred milliseconds [RCR11]*. Even more challenging, recognizing human voices takes 4 ms [Agu+10]*. These tasks are also computationally intensive and hence require to offload the processing. Ha et al. [Ha+14]* have presented a system for wearable cognitive assistance. The system uses Google Glass to capture live video and performs real-time scene interpretation using different components, such as activity inference, face recognition, and motion classifiers.

4.3 Summary

Table 4.1 summarizes prominent use cases from the previous Section 4.2 and shows how the promised benefits of Edge Computing (see Section 3.1) are applicable to them. For this, we use the following semantics:

++ Edge Computing is absolutely necessary to ensure the requirements, and these cannot be fulfilled by relying on the cloud. Furthermore, local processing cannot ensure the expected quality of experience.

+ Edge Computing is beneficial and improves the quality of the service and/or its experience for the users.

∘ The benefit of Edge Computing heavily depends on the concrete scenario and context in which the application operates.

– Edge Computing brings no real-world benefit or the attribute is not relevant for the application. For example, even though Edge Computing might improve the latency in absolute numbers, this might not be critical in applications where the computation or actuation takes far longer than the communication.

The last four columns of the table indicate which of the components as defined in Section 4.1.1 are used by that use case.

Table 4.1: Systematic overview of surveyed use cases[†]

	Latency	Bandwidth	Energy	Privacy	Consolidation	Filtering	Storage	Offloading
Mobile Device Augmentation								
Gaming								
Scene rendering [MKB18b; LS17]	+ +	+	+	–	✗	✗	✗	✓
Collaboration of neighboring players [Cai+18; PS18]	+ +	+	+	–	✓	✗	✓	✓
AR/VR								
Rendering [Shi+19]	+ +	+ +	+	–	✗	✓	✗	✓
Hybrid rendering [Lai+17]	+ +	+	+	–	✗	✓	✓	✓
Reconstruction of 3D maps [Boh+15]	+	+ +	+	–	✓	✓	✓	✓
Content delivery								
Video streams [AD14]	–	+	–	–	✗	✓	✓	✗
Website delivery [Zhu+13]	–	∘	–	–	✗	✗	✓	✗
Applications and updates [Bha+15b; Bha+15a]	–	+	–	–	✗	✗	✓	✗
Collaborative caching [LYS16; Tra+17]	+	–	∘	∘	✗	✓	✗	✗
Storage								
Storage for edge analytics [LMB17]	+	+	∘	∘	✓	✓	✓	✓
Reverse CDN [Sch+17; Ged+18b] [Psa+18; MSM17]	∘	+	∘	+	✗	✗	✓	✗
Document synchronization [Hao+17]	–	+	–	∘	✓	✗	✓	✗
Personal data storage [Cha+15; Mor+16; Per+17b]	∘	∘	∘	+	✗	✗	✓	✗
Infrastructure Augmentation								
Smart Grids								
Monitoring and control [Fre+13]	∘	+	∘	+	✓	✓	✓	✗
Scheduling distributed energy resources [PSM10]	+	∘	–	∘	✓	✗	✗	✓
Traffic & Transportation								
Adaptive traffic light [Gha+16]	–	∘	–	+	✓	✓	✗	✓
Detection of road hazards [CDO19]	∘	∘	–	–	✓	✓	✗	✓
Traffic planning [Zhe+11; Sat+17; QKB17]	–	∘	–	∘	✓	✓	✓	✗
Emergency vehicle route clearance [Nun+15]	–	+	–	–	✓	✗	✗	✗
Autonomous Driving								
Disseminating data to vehicle [DBH15; Yua+18]	+ +	+	–	∘	✓	✓	✗	✗
Processing LIDAR data [Qiu+18]	+	+ +	–	–	✗	✓	✗	✓

	Latency	Bandwidth	Energy	Privacy	Consolidation	Filtering	Storage	Offloading
Environment								
Pollution monitoring [ZLH13; Zhe+14b]	–	∘	–	–	✓	✓	✓	✗
Pollution monitoring via crowdsensing [Mai+09] [Sch+12; Dut+17; MAŽ18]	–	+	+	+	✓	✓	✓	✗
Optimizing garbage collection [AZM15; Per+14b]; [Aaz+16; Med+15]	–	–	–	–	✓	✓	✓	✗
Emergency Response								
Emergency notification [AH15]	–	∘	–	–	✗	✓	✗	✗
Situation awareness/ mobile command and control [Chu+13]	∘	+	–	∘	✓	✓	✓	✗
Ad-hoc communication in disaster scenarios [Meu+17c] [Sat+13]	+ +	+ +	–	–	✓	✓	✓	✗
Surveillance								
Vehicle tracking [Che+17a]	∘	+	–	+	✗	✓	✗	✓
Just-in-time video indexing [Sat+17]	+	+ +	∘	+	✗	✓	✓	✓
Biometric identification [Sto12]	–	∘	∘	+	✗	✗	✓	✓
IoT Device Augmentation								
Smart Home/Building								
Video surveillance [San+14] [AXS17]	–	+ +	–	+	✗	✓	✓	✗
Coordination of building subsystems [Fer+18]	∘	∘	–	+	✓	✓	✗	✗
Industrial IoT								
Production process analysis [AZH18; LGS17; Fu+18]	–	∘	∘	+	✓	✓	✓	✗
Machine condition monitoring [Wu+17a; Oye17]	∘	+	–	+	✓	✓	✗	✗
Warehouse logistics scheduling [LY18]	∘	+	–	+	✓	✓	✗	✗
Dynamic production line scheduling [Wan+18a]	+	∘	–	+	✓	✗	✗	✓
Agriculture & Farming								
Monitoring plants/lifestock [ADH18; Car+17]	–	+ +	+	∘	✓	✓	✗	✗
Yield prediction [Lia+18]	–	∘	–	+	✓	✗	✓	✓
Human Augmentation								
Quantified Self								
Analyzing fitness tracker data [Sch+15; Baj+15]	–	∘	+	+	✗	✓	✓	✓

	Latency	Bandwidth	Energy	Privacy	Consolidation	Filtering	Storage	Offloading
Precision Medicine								
Fall detection [Cao+15]	–	+	+	+	✗	✓	✗	✓
Patient monitoring with WBAN [AS16]	∘	+	∘	+	✓	✓	✓	✓
Remote surgery [Xu+14]	+	+	–	∘	✗	✓	✗	✗
Analyzing ECG features [AG10]	–	∘	–	+	✗	✓	✓	✓
Cognitive Assistance								
Face recognition [RCR11]	+	+	∘	+	✗	✗	✗	✓
Speech recognition [Agu+10]	+ +	∘	∘	+	✗	✗	✗	✓
Wearable cognitive assistance [Ha+14]	+ +	+	+	+	✓	✓	✗	✓

From this analysis, we conclude this chapter by summarizing the main observations:

Observation I: Diverse objectives | The motivations to use Edge Computing are very diverse. Consequently, most use cases only profit from a subset of the potential benefits. The heterogeneity of objectives and use cases means that there is no "one size fits all" solution when it comes to the question if an application should be moved to the edge.

Observation II: Not indispensable for most applications | While most of the presented applications can benefit from one or more aspects of Edge Computing, resulting in higher quality of service or user satisfaction, few applications cannot fundamentally function without Edge Computing.

Thus, economical considerations are important to determine if Edge Computing is sensible for a given use case. The fact that there are no established business models and ubiquitous infrastructures for Edge Computing yet prevents most of these applications from being moved to the edge today. However, once Edge Computing infrastructures are widely available, a large number of applications are likely to use it.

Observation III: "Killer apps" do exist | We identified some applications for which Edge Computing is indispensable, either regarding latency (e.g., rendering for AR/VR) or bandwidth (e.g., the processing of several video streams or LIDAR data). Furthermore, Edge Computing can provide an emergency communication and computing infrastructure, thus creating a more resilient overall public infrastructure.

Observation IV: Cross-layer application design | Many works propose layered system architectures, where applications are split between the layers. While the number and naming of the layers differ (the common ones being: local, edge, cloud, and sometimes intermediate fog layers), the common motivation is to exploit the favorable characteristics of each layer. How to efficiently partition applications across layers is still an ongoing field of research.

Observation V: Offloading objectives | There are two common motivations for offloading. First, offloading can accelerate latency-critical computations on devices with low computing power. Latency reductions are especially useful for demanding real-time computations, especially for mobile gaming, AR, VR, and autonomous driving. The second motivation for offloading is to save energy on battery-constrained devices. Furthermore, energy benefits are usually only viewed from the perspective of the end devices, and the effect on the overall energy footprint remains unclear, e.g., when taking into account the energy efficiency of the edge surrogates or network middleboxes.

Observation VI: Bandwidth is critical for infrastructure augmentation and IoT devices | Applications in the domain of infrastructure augmentation and IoT devices tend to have little demand for computation offloading. However, nearly all of our surveyed applications in these two categories implement the consolidation and filtering components. Compared to the cloud, Edge Computing can achieve large bandwidth savings for applications that process big ephemeral data. Thus, while latency tends to be only a minor issue in those use cases, they can profit from the bandwidth-saving aspect of Edge Computing.

Observation VII: Privacy has great potential | There is a clear division between applications where the privacy-protecting aspect of Edge Computing is relevant and those where it is not. The privacy aspect can be of tremendous relevance for sensor data that contains trade secrets or sensitive personal data. There is great untapped potential for research in this direction to fully exploit the privacy benefits of Edge Computing.

4.4 Conclusion and Requirements

In chapters 2–4, we have provided a detailed description and analysis of the field of Edge Computing, starting from a taxonomy (Chapter 2), analyzing its characteristics (Chapter 3), and providing a detailed survey of Edge Computing applications (Chapter 4). From these findings, this chapter derives requirements for Urban Edge Computing and outlines how those are addressed in the remainder of this thesis.

Requirement I (RQ-1): Dense network of micro data centers | Existing data center infrastructures will not be able to fulfill the requirement of providing proximate computing resources. This is especially true if we consider mobile users in an urban area. Those users have demanding requirements in terms of resources and mobility (as they change their position frequently). To provide good quality of service to those users, we need new types of dense cloudlet infrastructures in our surroundings.

Requirement II (RQ-2): Dynamic composition of applications | In most cases, it is beneficial to offload only certain parts of an application, e.g., those that are the most compute-intensive or have the most impact on the device's battery life. Hence, to enable fine-grained offloading of application functionality, edge-enabled applications should be composed of modular parts. Following the paradigm of microservices, these parts can be dynamically composed and offloaded at runtime.

Requirement III (RQ-3): Low-overhead offloading | Offloading applications or part thereof incurs an overhead, e.g., by having to transfer the offloadable parts from a (mobile) device to the surrogate. Ideally, this overhead should be low, both to save energy and bandwidth. This can, for instance, be achieved by pre-provisioning or caching offloaded services.

Requirement IV (RQ-4): Sharing of services at runtime | While Cloud Computing infrastructures are able to offer highly scalable and elastic resources, this is more challenging in an edge environment because the resources at individual cloudlet locations are limited. Based on the observation that we operate in a highly dynamic environment (with regards to changes in demands, user locations, and network conditions) and resources at a single edge location are limited, running service instances should be shared among different client applications. To realize this, services should be implemented as standardized and reusable components. Sharing services avoids over- and underprovisioning of resources, and hence, allows for a more efficient usage of resources.

Requirement V (RQ-5): Sophisticated placement decision for computing functionalities and data | The dynamic composition of (shared) applications means that for each component, we need to make an informed decision—e.g., based on available resources, data location, and network connections—on where to place it on available (edge) resources. In addition to functional components, data captured by (mobile) users should be placed in a way that takes into account the user's context and facilitates sharing.

Requirement VI (RQ-6): Dynamic service adaptations at runtime | A single functionality can be implemented in a number of ways, with varying impact on the quality of the computation and the resulting runtime of the service. In a dynamic edge environment, where pipelined services are shared among different applications, services should be adaptable to tradeoff those characteristics to optimally meet users' demands. Besides the integration of mechanisms that control the execution of service variants, this also requires support for application developers for the definition and assessment of service variants.

4.4.1 Remaining Thesis Outline

The contributions of parts II–IV are structured following a bottom-up approach and address the requirements defined above, as shown in Figure 4.3. We begin with the infrastructure (Part II), followed by the execution framework for Edge Computing (Part III) and, lastly, the strategies and runtime adaptations that can be applied during the system's execution (Part IV).

Chapters 5 and 6 address RQ-1 by investigating the deployment of cloudlets on a city-scale infrastructure. The Edge Computing framework presented in Chapter 7 is based on the paradigm of independent and composable microservices (RQ-2). Provisioning of services is done through a microservice store, avoiding costly (prior) transfers of executables from the client devices to surrogates (RQ-3). The proposed Edge Computing framework also allows the sharing of running service instances (RQ-4).

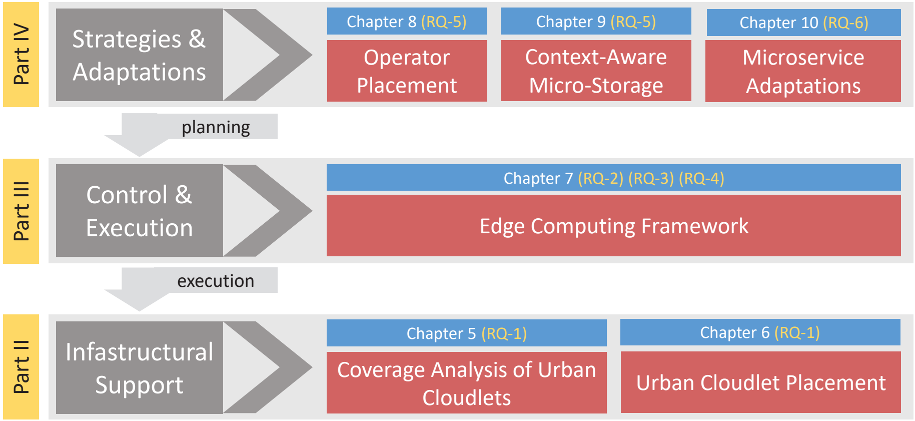
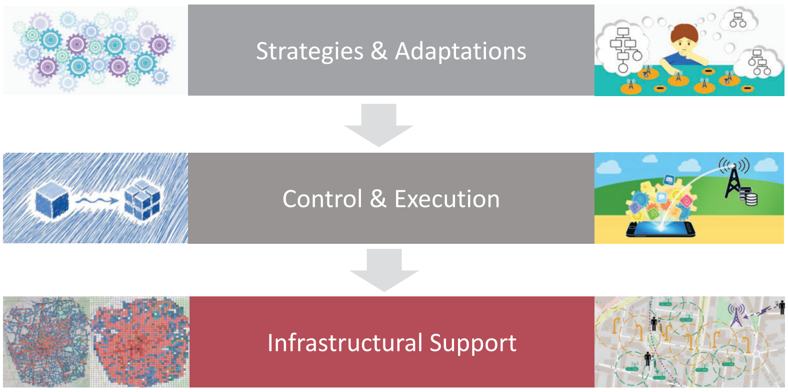


FIGURE 4.3: CONTRIBUTIONS OF PARTS II–IV

Chapters 8 and 9 address the placement of functional parts of applications and data (RQ-5). Lastly, Chapter 10 proposes mechanisms to make microservices adaptable at runtime (RQ-6).

Part II

Infrastructural Support



In this part, we investigate the physical infrastructural support required for Urban Edge Computing, i.e., the actual deployment environment of cloudlets in an urban area. The infrastructure presented in this part is the basis to provide Edge Computing services. The concepts presented in the subsequent parts of the thesis are deployed on top of this infrastructure. In detail, this part addresses two problems.

First—based on the observation that building new infrastructure from scratch is costly—Chapter 5 analyzes how we can leverage existing infrastructure resources in a city to place cloudlets. This chapter is largely based on the publications [Ged+18c; Ged+18d].

Second, given urban infrastructures for a *potential* placement of cloudlets, Chapter 6 investigates the problem of where to place cloudlets on heterogeneous infrastructures. This chapter in turn is in large parts based on the work published in [Ged+18d].

In this part, text segments that are verbatim copies of [Ged+18c] or [Ged+18d] are printed in *brown color* and *gray color*, respectively. Tables and figures taken or adapted from these publications are marked with † ([Ged+18d]) and †† ([Ged+18c]) in their caption.
Contribution statement: The student Jeff Krisztinkovics helped in the collection of access point data and implemented parts of the functionality for the computation of coverage in the context of his Bachelor's thesis [Kri19].

CHAPTER 5

Coverage Analysis of Urban Cloudlets

Chapter Outline

5.1 Introduction

One of the important building blocks and a recurring term used in the description of Edge Computing deployments is the concept of cloudlets—small-scale data centers that offer proximate resources for storage and computations (see Chapter 2). Cloudlets can therefore be considered as the infrastructural basis which our other contributions build upon. Most importantly, cloudlets will be responsible for hosting the execution and control environment for Edge Computing applications (see Chapter 7).

Previous research has addressed various problems that often relate to runtime issues of cloudlet deployments, for instance offloading mechanisms (e.g., [Cue+10]) and programming models (e.g., [Hon+13]). However, little attention has been paid to the question of where to deploy cloudlets on a city-scale. A dense deployment of cloudlets is one of the requirements we have identified for Urban Edge Computing (see RQ-1 in Section 4.4). Cloudlets require resources to be deployed and operated. Besides power and network connectivity, we require physical space to install (additional) hardware that serves as cloudlets. In this chapter, we refer to locations where such resources are available as *infrastructure*. Contrary to a data center environment, where the abovementioned resources are inherently present, this is not the case in an urban environment, making the deployment of cloudlets more challenging. Because newly building those resources is costly, this chapter investigates if we can leverage existing urban infrastructures for the placement of cloudlets and hence, provide *infrastructural support* for Edge Computing applications. More specifically, we suggest to co-locate cloudlets with wireless access points at three types of existing infrastructures present in cities: (i) cellular base stations, (ii) commercial off-the-shelf WiFi routers, and (iii) smart street lamps. We argue that those infrastructures are viable locations to deploy cloudlets. In Section 5.2, we describe them in more detail and study their particular characteristics.

After reviewing related work in Section 5.3, the following sections investigate whether these infrastructures are sufficient to provide coverage in an urban area, given that realistically, only a fraction of them will be upgraded to host cloudlets. To answer this question, we perform a systematic *coverage analysis* (Section 5.6), using four different *coverage metrics*, as defined in Section 5.5. The different coverage metrics allow us to quantify the quality of service that can be expected for different application use cases. In order to base our analysis on realistic data, we collected a large body of data (see Section 5.4) for a German city, containing locations of the infrastructures and user traces from mobile applications. The results of the coverage analysis based on these datasets serve two purposes:

- We are able to show that leveraging urban infrastructures has the potential to enable a city-wide deployment of cloudlets. A cloudlet deployment that covers large parts of a city is furthermore an important enabler for future smart city applications, like the ones presented in Section 4.2.2.
- The result of the analysis can serve as an aid for planning a cloudlet deployment, e.g., by estimating the number of cloudlets for a target coverage. Furthermore, it allows for the identification of regions without coverage where infrastructure resources must be deployed in order to provide Edge Computing services.

5.2 A Multi-Cloudlet Urban Environment

In this section, we present three types of infrastructures that are able to accommodate cloudlets. We assume that these infrastructures host wireless access points to which users can connect and hence, access the resources and services provided by the cloudlets. Before describing them in detail, we first describe the general characteristics and requirements of such infrastructures.

For our urban scenario, we consider cloudlets to be hosted on three types of infrastructure: cellular base stations, routers, and street lamps. We specifically investigate those types of infrastructure because they are ubiquitously present in urban spaces. Figure 5.1 illustrates an environment of different access point types, with cell towers in *purple*, routers in *green*, and smart street lamps colored in *orange*. Common to all these three types of access points is their dense deployment in urban spaces and their function as a wireless connectivity gateway to mobile users. For cellular base stations and routers, we can further assume a powerful backhaul connection in terms of bandwidth and ample physical space at the point of presence to co-locate additional hardware.

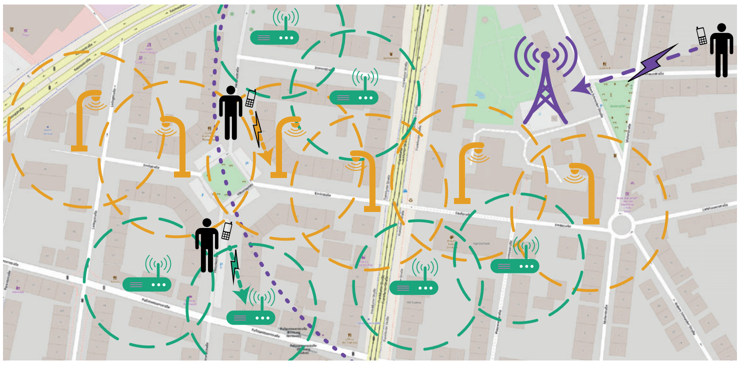


FIGURE 5.1: A MULTI-CLOUDLET URBAN INFRASTRUCTURE†

Mobile users in the vicinity of these cloudlets can then make use of them to offload data and computations. The different types of access points are heterogeneous in several ways. First, due to different wireless access technologies, their communication ranges vary. A cellular base station can cover a wider area than a WiFi router in an urban environment. With the advent of new communication technologies, we expect to see other kinds of deployments, e.g., small-scale cellular base stations (so-called femtocells [CAG08]) on street lamps. Second, due to the varying physical space available for hardware installations at the access points, the resulting computing power colocated at one access point will differ. For example, installing server-grade hardware is difficult in the restricted space of street lamps, while this is likely the predominant deployment model for cloudlets at the Radio Access Network, i.e., at the location of cell towers. Depending on the resources they provide and the underlying business model, installing cloudlets incurs varying costs. Chapter 6 will describe how this heterogeneity makes the decision challenging, which access points to equip with cloudlets.

Lastly, we have different stakeholders that own or operate the infrastructure, ranging from ISPs and mobile network operators to businesses, municipalities, or private citizens. We refer the reader to Section 3.4 and Section 11.2.3 for a more extensive discussion on stakeholders and (future) business models for Edge Computing. Regardless of the underlying business model for the usage of cloudlets, we argue that the use of existing infrastructure as well as future infrastructures, such as lamp posts in the context of smart cities, allows a cost-effective placement of cloudlets, since it avoids the construction of new access points. Table 5.1 summarizes the characteristics of each cloudlet type.

5.2.1 Cellular Base Stations

Each major city today features widespread cellular coverage, albeit at varying quality and speed. Even though some areas still lack satisfying coverage and connection speeds, cellular base stations represent a viable location for the deployment of cloudlets if we want to cover large areas. First, existing radio access networks have a high-bandwidth backlink on-site. In the case of cloudlets, this could be important if there is the need to retrieve large amounts of data from the cloud. Furthermore, this is an important asset to perform quick intra-network handover and enable distributed computations. Second, at most cellular stations, there is

TABLE 5.1: CHARACTERISTICS OF ACCESS POINT TYPES†

	Cellular base stations	**Routers**	**Street lamps**
Density	low	high	high
Ownership	mobile network provider	ISP, businesses or private	municipality
Access technology	3G, 4G, 5G	WiFi	WiFi, Femtocells, LoRa, mmWave
Communication range	high	low	low-medium
Computational resources	high	low–medium	low

enough physical space available to install massive computing resources in the form of server-grade hardware. Another advantage of cellular base stations is their high reliability [EZB17].

Leveraging resources co-located with the radio access network is commonly referred to as Mobile Edge Computing (MEC) and seen as a viable deployment model to make Edge Computing available [Li+16b; Abb+18; SBD18]. This deployment model is predicted to gain importance with the advent of the new 5G cellular standards [Nun+15; Hu+15b] because those offer processing capabilities *within* the network. For network operators and cellular service providers, Mobile Edge Computing is a future business opportunity, as they will be able to rent out computational resources located at base stations.

5.2.2 Routers

Next, we consider commercial off-the-shelf WiFi routers. In urban areas, the density of WiFi routers is very high. This includes both privately-owned devices as well as public access points offered by businesses like cafes or restaurants. The latter is a service increasingly valued by customers. Using WiFi routers as cloudlets is advantageous, given their ubiquity. Performing computations on the routers themselves has been investigated before (see Section 5.3.1), but they also offer the possibility to co-locate rather powerful hardware in the local network they are connected to.

While the motivation for businesses to provide cloudlets might be to enhance service for their customers, the incentives for private citizens are less obvious. We can, however, make two observations in this regard. First, several initiatives already promote the sharing of ones WiFi to give others internet access (e.g., Freifunk[1] in Germany). We argue that going one step further—from providing free access to free computations and/or storage—is the next logical step. Second, the incentive to provide a cloudlet could also come from one's internet service provider. For instance, service providers could install equipment and applications at the users' premises and compensate them with a reduced subscription bill.

[1] https://freifunk.net/ (accessed: 2019-10-30)

5.2.3 Street Lamps

Besides service providers, businesses, and private citizens, municipalities also have an inherent interest in enabling services that lead to smarter cities. For this reason, we envision cloudlets to be placed on lamp posts. A lamp post is a viable location to place a cloudlet for two reasons: First, there is a large number deployed in every city, sometimes with the distance between two lamp posts being only a couple of meters. Therefore, especially in densely populated areas, they can very well complement cell towers and routers to provide dense coverage. Second, from the perspective of users moving on a city street, the wireless signal is less obstructed compared to WiFi access points that are typically placed in buildings.

Upgrading lamp posts to host cloudlets might seem to incur a huge investment at first, since wireless access points and uplink networks are typically not readily available. However, we can observe a trend that municipalities around the world are currently in the process of updating their street lighting, mostly due to energy considerations. According to the *Humble Lamppost* project[2], 75 % of lamp posts in Europe are over 25 years old and consume between 20 % and 50 % of a city's energy budget. Therefore, investments to upgrade lamp posts to LED-based lighting will quickly amortize due to the energy savings of LED lamps. In the process of upgrading, additional functionalities, such as sensory capabilities, network connectivity, and computing resources can be installed to turn traditional street lamps into *smart* street lamps. A number of commercial products are already available, e.g., the SM!GHT[3] lamp by the German company *EnBW*. These efforts towards *smart street lamps* furthermore have resulted in a DIN standard [DIN18].

We argue that in view of this trend, installing additional hardware to provide computing resources is a negligible additional investment, compared to the overall cost of upgrading the street lamps. It should however be noted, that upgrading street lamps might also require changes in the entire electric infrastructure that powers the lamps. For instance, today most lamps do not feature a switch that could control the power supply of different components. Instead, lamps are centrally turned on and off.

In conclusion, this section presented three types of available infrastructures in today's cities that meet the requirements to be equipped with cloudlets: dense deployment, physical space and power supply for additional hardware, and wireless network connectivity.

5.3 Related Work

5.3.1 Urban Cloudlets

To the best of our knowledge, previous works have not considered the combined use of different types of cloudlet infrastructures in an urban area. Providing offloading capabilities at the radio access network (RAN) has been investigated in a number of publications [Bec+14; Mao+17; Abb+18], with the special case of so-called femtocells [CAG08], which are less expensive to deploy and operate. WiFi routers have been suggested as Edge Computing devices, either as hosts for the computations themselves [Meu+15], as a joint infrastructure for computations and

[2] https://eu-smartcities.eu/initiatives/78/description (accessed: 2019-10-30)
[3] https://smight.com/en/ (accessed: 2019-10-30)

bridging communications [Meu+17c], or to perform auxiliary functions in an Edge Computing system, e.g., the discovery of Edge Computing nodes [Ged+17].

Some works [BMV10; Lee+13] investigate the interplay between WiFi and cellular connections. Balasubramanian, Mahajan, and Venkataramani [BMV10] study the availability of 3G and WiFi networks in a city and suggest two different switching mechanisms—for delay-tolerant and delay-sensitive applications— to complement 3G connections with WiFi networks. In an empirical study, the authors compare throughput and loss rates for both network types and argue that WiFi networks can help to reduce costs and alleviate stress on cellular networks. Lee et al. [Lee+13] conduct a quantitative study on the savings of traffic and battery power when offloading 3G traffic through WiFi. The authors also consider applications where long delays (in the order of 100 seconds) are tolerable. However, as we have argued in Section 3.1 and Chapter 4, these are not realistic assumptions for applications that will be deployed at the edge. Similarly, Dimatteo et al. [Dim+11] have shown that a small number of WiFi access points are sufficient to serve delay-tolerant applications.

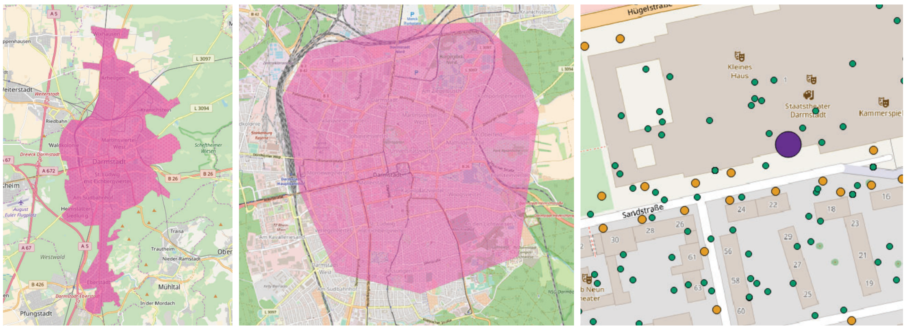
Visions for offloading infrastructures also include the use of vehicles [Wan+16; GZG13; Hou+16] and drones to host cloudlets [Sat+16; JSK18a]. While some works have proposed to make existing street lamps smarter [Jia+18], to the best of our knowledge, our contributions in [Ged+18c; Ged+18d] were the first to consider street lamps for hosting cloudlets.

5.3.2 Coverage

The term *coverage* is most widely used in the context of Wireless Sensor Networks (WSNs) [HT05]. Consequently, the problem of coverage has been studied extensively in this context, as analyzed in various surveys [Wan11; MHS17; MA10; GD08]. In WSNs, coverage describes how well an area of interest can be monitored [MHS17; Wan11] by sensors or people [GS15]. Coverage is therefore an aspect and a metric for the quality of service the network of sensors is able to deliver.

Several established definitions of coverage exist, for example, sweep coverage [Li+11] or barrier coverage [Liu+08]. However, many of those definitions cannot meaningfully be applied to analyze cloudlet coverage in an urban scenario. For instance, barrier coverage denotes the singular detection of a target inside an area. This is not a sensible metric for our application domain because we want mobile users to have a continuous connection to cloudlets and not just at a single point in time. Close to our definitions of coverage that will be introduced in Section 5.5, the work of Fan et al. [FJ10] proposes three different definitions of coverage, namely area, point, and path coverage. We adapt those to our urban cloudlet context and will add *time coverage* as another metric for coverage. In our urban cloudlet scenario, coverage indicates the connection to an access point on which a cloudlet is co-located, and hence, is a metric for how well offloading demands of mobile users can be served by cloudlets.

A very limited number of works have investigated coverage in the context of cloudlet deployments for Edge Computing. Syamkumar et al. [SBD18] analyze the deployment characteristics of cell towers w.r.t. the population distribution and density in the US. Other works have investigated the coverage of WiFi access points in case studies related to computational offloading, but only considered one type of coverage, e.g., spatial [Bur+15], point [Mot+13], or temporal [Meu+17a] coverage. As an exception, Lee et al. [Lee+13] consider both spatial and temporal



(a) Administrative city boundary†

(b) Inner city area†

(c) Access point locations on a sample section of the city center

FIGURE 5.2: DARMSTADT CITY CLOUDLET DATASET

coverage for WiFi access points. Similar to what we describe in the related work about the cloudlet placement problem (see Section 6.2), Liao et al. [Lia+11] assume that we can freely place cloudlets to optimize their coverage. In contrast, this chapter is intended to provide an empirical study on the coverage of *existing* access point infrastructure.

To conclude the analysis of related work, none of the existing works have jointly considered the three types of cloudlet infrastructure (cellular base stations, routers, street lamps), both for the coverage analysis with four types of coverage (spatial, point, path, and time coverage) and the placement of cloudlets on those infrastructures.

5.4 Datasets

We investigate the placement of cloudlets in the city of Darmstadt (Germany), a major city with a population of about 150 000. To do so, we use real-world data for both the location of access points and the traces of mobile users as described hereinafter. The official administrative boundary of the city is depicted in Figure 5.2(a). We restrict our analysis in the remainder of the paper to the inner city area (spanning an area of 15.57 km^2) as shown in Figure 5.2(b) because most of the access point data gathered lies within that boundary. This is especially true for the routers, which were collected by volunteers. Furthermore, the inner city area allows us to study the interplay between all three types of infrastructure, not all of which might be available with the same density in more rural areas.

5.4.1 Access Point Locations

In total, we collected the locations of nearly 50 000 access points throughout the city for the different types of access points. Figure 5.2(c) shows a typical distribution of cell towers (purple), routers (green), and lamp posts (orange) on an exemplary small section of Darmstadt. We now describe the origin and extent of the access point location data.

5.4.1.a Cellular Base Stations

The *Bundesnetzagentur* (Federal Network Agency) is the regulating body in Germany in charge of authorizing and supervising the operation of radio installations. A map of all transmitting stations, including cell towers, can be accessed through their website[4]. However, the website does not provide a feature to export the data. Thus, we performed a manual crawl using the network panel of the Google Chrome browser developer tool. We issued a query of all the cell towers within the city and parsed the resulting JSON data that contains their GPS locations.

5.4.1.b WiFi Routers

We followed a so-called wardriving approach to collect information about WiFi networks in the city and used this data to estimate the position of routers within the city. Using a modified version of *WiFiAnalyzer*[5], an open source Android application, volunteers walked around the city and collected the signals from available WiFi access points. We used the raw data from two volunteering campaigns, conducted in March 2016 and February 2018. In total, 27 participants—mostly students—were involved. We made the dataset and the source code of the tools publicly available to the research community[6]. More details about the collection process are given in Explanation 5.1. It is important to note that this dataset might include some wrong or inaccurate data. This is due to the nature of the wardriving approach. GPS receivers only operate with a certain accuracy and calculating the distance to an access point from the received signal strength is based on an approximative model. However, we argue that overall, this gives a reasonable estimation of the available routers to place cloudlets. More importantly, the data was collected while walking or driving through the city and not inside buildings or private locations, therefore reflecting the usage context of a mobile user who wishes to perform opportunistic offloading.

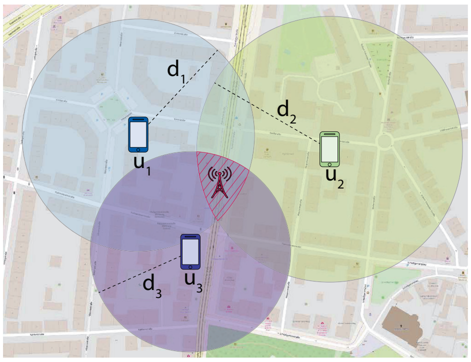
> EXPLANATION 5.1: DATA COLLECTION THROUGH WARDRIVING
>
> Wardriving is "*the act of moving around a specific area and mapping the population of wireless access points for statistical purposes*" [Hur04]. We obtained raw measurements from the aforementioned Android application, containing signaling information from the access point, such as their SSID and BSSID. Furthermore, based on the received signal strength indicator (RSSI), the application estimates the user's distance to the access point. We collect this raw measurement data from every participant. Before further processing, we perform a lookup on the router's MAC address, i.e., the reported BSSID, and manually eliminate all entries from manufacturers that do not produce routers. The positions of the access points were then estimated via multilateration from multiple measurements of the same access point.

[4] http://emf3.bundesnetzagentur.de/karte/ (accessed: 2019-11-16)

[5] https://github.com/VREMSoftwareDevelopment/WiFiAnalyzer (accessed: 2019-11-01)

[6] https://github.com/Telecooperation/darmstadt-wifi (accessed: 2019-11-07)

> For multilateration, a minimum of 3 measurements is required (trilateration). The figure on the right illustrates this for 3 users u_1, u_2, u_3 that report distances d_1, d_2, d_3 to the access point, with the possible location of the access point being in the hatched red area. We use an iterative approximation algorithm, implemented in Ruby, to find a valid location, i.e., one where the circles representing the distances intersect. The relevant part of the source code can be found in Appendix A.



5.4.1.c Street Lamps

We obtained a database export of the position of all street lighting installations in Darmstadt from *e-netz südhessen GmbH*[7], the company in charge of managing the city's electrical infrastructure. The dataset includes different types of street lighting, such as lights hung via cables over streets, but we only include fixed lamp posts for our further analysis of cloudlet coverage and placement, as they provide enough space and a safe enclosure to install additional hardware for cloudlets.

To conclude the description of the access point datasets, Table 5.2 summarizes the number and density of each access point type for the entire administrative city boundary and the inner city area (see Figures 5.2(a) and 5.2(b)).

TABLE 5.2: NUMBER OF COLLECTED ACCESS POINTS†

	Cellular base stations	**Routers**	**Street lamps**
Administrative boundary	205	34 699	14 331
Density per km^2	1.7	284.0	117.3
Inner city	66	31 974	5608
Density per km^2	4.5	2194.5	384.9

5.4.2 Mobility Traces

Several important types of coverage take into account the user's position. Therefore, we need realistic mobility traces that reflect where in the city we have demands for offloading. While a lot of research exists about mobility *models* [Zhu+15; Jar+03; YLN03], we want to base most parts of our analysis on *real-world* mobility traces.

[7] https://www.e-netz-suedhessen.de/ (accessed: 2019-11-07)

TABLE 5.3: MOBILE APPLICATION TRACES†

	Kraken.me	Ingress	CrowdSenSim
Users	205	1401	2499
Data points	437 417	520 409	431 001
Paths	11 930	47 915	44 150

For our analysis, we use data from two mobile applications, *Kraken.me* and *Ingress*, described in Sections 5.4.2.a and 5.4.2.b. The former is a persona tracking framework while the latter is a mobile AR game. The datasets allow us to model where offloading capacities will be required in the future. For instance, upcoming versions of the Ingress game might require more sophisticated processing for AR that cannot be handled by the mobile device itself. Additionally, we include simulated mobility data generated with *CrowdSenSim* (see Section 5.4.2.c), a tool designed for simulating crowdsensing applications.

The three datasets differ with respect to the mobility patterns they represent. In addition, they feature locations both inside buildings and outside. We believe that combining them in our analysis allows our findings to be applied to more than one application use case. For example, Kraken.me records the daily activities of users, i.e., for a large fraction of time users are at home or work, while Ingress directs users to specific locations in the city.

From the raw data, we first perform *trace analysis* [Pan+13a] in order to obtain meaningful representations for the analysis (e.g., paths with sufficient lengths) and to reduce redundant data (e.g., data points that are very close to each other). In addition, this step also removes data points that are obviously erroneous. For example, some user devices in the Kraken.me applications reported wrong positions (outside Germany although the campaign took place only there) and timestamps (outside of the campaign duration). Further details and criteria for filtering the raw data are described in Explanation 5.2. Table 5.3 summarizes the resulting number of distinct users, data points, and paths after this pre-processing.

EXPLANATION 5.2: LOCATION DATA PRE-PROCESSING

We apply the following pre-processing steps to the raw location data obtained from our datasets:

- Data points not contained within the inner city area of Darmstadt (see Figure 5.2(b)) are discarded.
- We remove data points with invalid timestamps, i.e., locations that are timestamped outside the range of when the measurements are known to have happened. Not doing so would lead to a wrong construction of paths.
- In the case when the user has not moved between two consecutive (based on the timestamps) points, we remove the second data point.
- We define a *speed threshold* as the maximum plausible travel speed between two points. This serves to remove unrealistic (in terms of travel speed) path segments.

- To reduce the overall size of the datasets (and hence, make the computations on them more efficient) we introduce a *distance threshold*. Especially in datasets that have a high temporal resolution and where users are not constantly on the move, many consecutive data points will only slightly differ in their position. If the distance between two consecutive data points is below this threshold, the second point is removed.

- To analyze the coverage of entire *paths* rather than individual data points, we need to aggregate a series of data points to a path. We define the *time threshold* as the maximum time difference that two consecutive points can have if they are part of one path. For greater time differences, a new path is constructed. This mechanism therefore does not remove any data points but is used to generate paths from data points. Without this preprocessing step, all data points from one user would belong to one path only. Instead, the time threshold allows to model a continuous activity from the user (e.g., traveling between points of interest) during which we want to measure the cloudlet coverage.

- For the same reason, if the number of data points per path is below a certain minimum, the entire path is discarded.

- We define the *minimum path extension* for each path as the minimum area of the path's bounding box. This is done to eliminate paths with insufficient spatial extension (e.g., users who are circulating in a small area in or around their homes) that would distort the analysis, especially when comparing different coverage metrics. The minimum path extension therefore allows discarding user activity that is not relevant to the coverage analysis, e.g., when a user is staying within the boundaries of his home environment and therefore covered by the home access point.

5.4.2.a Kraken.me

Kraken.me [SS14] is a tracking framework that records users' activities and gathers data from various soft and hard sensors on mobile devices in order to provide personal assistance. During the development, a user study was conducted for several weeks using Android phones. Participants of the study were mostly students and university research staff. For our evaluations, we use a reduced dataset that only contains the timestamped positions along with a unique user ID. The data is temporally fine-grained with user positions being reported every 30 to 60 seconds on average.

5.4.2.b Ingress

Ingress[8] is a popular mobile AR game and the predecessor of *Pokémon Go*. Players visit portals at physical locations in the city. Portals are located at places of interest throughout the city, e.g., prominent buildings, landmarks, or transport stations. Each player needs to visit and interact with multiple portals, which leads to a constant movement of the player in the real world. Consequently, the users' positions are recorded implicitly by their interaction at the portals. In total, there are 724

[8] https://www.ingress.com/ (accessed: 2019-11-05)

portals located in the inner city area of Darmstadt. The current state of the game and player activity is visible on the Ingress Intel Map website[9].

We used game data gathered by a crawling tool provided by the authors of [Fel+18]. The crawler uses an automation tool for web browsers to request changes in the game state every second. It is important to note that changes include the position updates from players at portals. Because the user locations are only recorded at the portals and not between, the data is more coarse-grained in terms of temporal resolution compared to the Kraken.me data. However, the positions are still a good indicator for offloading demands related to other applications, since portals are often located at points of interest in the city.

5.4.2.c CrowdSenSim

Lastly, to extend the number of available data points for our analysis, we use artificial mobility traces generated with *CrowdSenSim* [Fia+17], a discrete-event simulator for mobile crowdsensing. CrowdSenSim generates user traces in urban areas where users roam around the city and randomly take turns onto streets. The simulator uses a uniform mobility algorithm[10] We set the simulation parameters such that several simulations are carried out for 7 days with 2500 users. The minimum and maximum travel times per path were set to 30 minutes and 720 minutes, respectively.

5.5 Coverage Metrics

For our analysis of urban cloudlet coverage, we consider four different metrics for coverage: spatial, point, path, and time coverage. These metrics are depicted in Figure 5.3[11] and differ in the data they require to be assessed. Spatial coverage is completely independent of individual user locations. Point coverage requires a set of user locations and path coverage an (implicit or explicit) ordering of those. Point and path coverage allow to more accurately model actual demand patterns because users are not uniformly distributed throughout an area. Additionally, to compute the time coverage, user locations must include a timestamp at each location. These different coverage metrics allow the modeling of different offloading requirements, depending on the applications and user demands. For instance, point coverage can model how well singular offloading demands at certain locations can be served, while path and time coverage model applications that require continuous (with respect to distance and time traveled) availability of services offered by cloudlets.

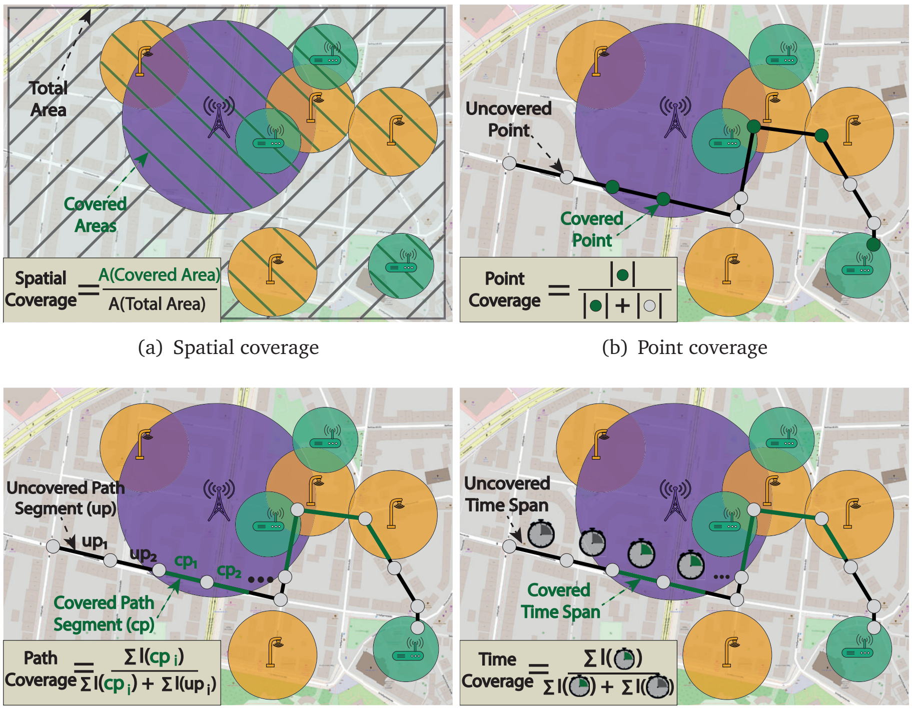
5.5.1 Spatial Coverage

Spatial coverage quantifies the spatial extent of the cloudlets' communication ranges in relation to the total area. Consequently, spatial coverage gives only an indication of how well an area is covered by cloudlets and does not consider user locations. It is defined as the ratio between the union of the communication ranges

[9] https://www.ingress.com/intel (accessed: 2019-11-05)

[10] details can be found in the manual of the simulator: https://crowdsensim.gforge.uni.lu/ftp/manual1.1.pdf (accessed: 2020-05-17)

[11] the communication ranges in the figure are illustrative and not drawn to scale


Total Area
Covered Areas
Spatial Coverage = A(Covered Area) / A(Total Area)
Uncovered Point
Covered Point
Point Coverage = |●| / |●| + |○|
Uncovered Path Segment (up)
up1
up2
cp1
cp2
Covered Path Segment (cp)
Path Coverage = Σ I(cp i) / Σ I(cp i) + Σ I(up i)
Uncovered Time Span
Covered Time Span
Time Coverage

(a) Spatial coverage

(b) Point coverage

(c) Path coverage

(d) Time coverage

FIGURE 5.3: COVERAGE METRICS†

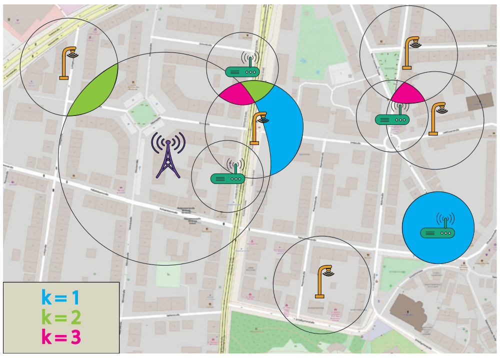
of cloudlets and the total size of the area, as shown in Figure 5.3(a). Formally, given a function A that represents the area, spatial coverage can be defined as

$$Spatial\ Coverage = \frac{A(covered\ area)}{A(total\ area)} \tag{5.1}$$

5.5.1.a *k*-coverage

Spatial coverage can be generalized to k-coverage. Instead of only considering if an area is covered by the range of one access point, k-coverage models the simultaneous coverage by k access points. Hence, we define k-coverage as follows: An area is said to be *k-covered* iff it intersects with the communication ranges of at least $k-1$ other access points. The reason one might want to expand the analysis to $k > 1$ is to explore the possibility of choosing between multiple cloudlets to optimize user experience in terms of connection bandwidth or resources. This choice could for instance be based on application requirements, e.g., a real-time application would choose the cloudlet with the lowest overall latency. As another example, in areas where many users are present, one cloudlet might not be sufficient to satisfy all offloading demands of users within its range.

Explanation 5.3: Spatial k-coverage



Following this definition, our previously introduced example of spatial coverage would equal to $k = 1$. This figure illustrates examples for $k = 1$, $k = 2$, and $k = 3$ at different intersections of the cloudlets' communication ranges.

5.5.2 Point Coverage

Point coverage indicates how many recorded location points of a mobile user are within the communication range of a cloudlet, as depicted in Figure 5.3(b). This coverage metric can therefore be used to model how many user requests at the captured distinct points can be served by a cloudlet. Given a set CP of covered points and a set UP of uncovered points, point coverage is defined as

$$Point\ Coverage = \frac{|CP|}{|CP| + |UP|} \tag{5.2}$$

5.5.3 Path Coverage

Since users also move between the distinct points at which their position is recorded, path coverage (see Figure 5.3(c)) takes into account the paths of users. This allows us to model use cases where users need continuous connectivity to a cloudlet, e.g., when continuously processing video streams. By segmenting the entire paths into parts that are covered and uncovered, path coverage is defined as the ratio between the length of covered segments and the total path length. Because the path is not explicitly recorded but constructed with individual user locations (e.g., captured periodically by a phone's GPS receiver), path coverage makes the assumption that users move on a straight line between two subsequent locations. As a further simplification, we only consider segments between two covered points. From the previous definition of uncovered points and covered points, we therefore obtain n covered path segments (cp_i) and m uncovered path segments (up_j). Given l as a function for the length of a segment, we can hence define path coverage as

$$Path\ Coverage = \frac{\sum_{i=1}^{n} l(cp_i)}{\sum_{i=1}^{n} l(cp_i) + \sum_{j=1}^{m} l(up_j)} \tag{5.3}$$

5.5.4 Time Coverage

On their paths, users might travel at different speeds, and thus, might be on a path segment for different durations. Time coverage (sometimes also referred to as *temporal coverage*) takes this into account, as shown in Figure 5.3(c). This metric works in a similar way as path coverage, but instead of the length of the path segments considers their duration. Time coverage is therefore defined as the ratio between the total time a user is on a connected path segment and the total travel time of the path. This metric allows to model how long users can be connected to a cloudlet and therefore—similar to path coverage—can represent the perceived quality of the service in a more fine-grained way. Given the previous definition of covered and uncovered segments, and a function t that computes the duration path segments, time coverage is defined as

$$Time\ Coverage = \frac{\sum_{i=1}^{n} t(cp_i)}{\sum_{i=1}^{n} t(cp_i) + \sum_{j=1}^{m} t(up_j)} \tag{5.4}$$

5.6 Coverage Analysis

We now analyze the coverage of urban cloudlets according to the metrics defined in Section 5.5 and by using the datasets presented in Section 5.4.

5.6.1 Methodology

Development environment and setup: We import the datasets into a PostgreSQL[12] relational database. The database system uses the PostGIS extension[13] that enables the representation of spatial data and provides functions that operate on spatial data (e.g., computing the intersection or union of shapes). We use several scripts for the data pre-processing and the computation of coverage. The scripts are written in the Ruby programming language, with some auxiliary functions implemented using the PL/pgSQL programming language. For the pre-processing of the raw data (see Explanation 5.2), we use the parameters as shown in Table 5.4. We base all following analyses on this pre-processed subset.

Evaluation scenarios: To reflect different deployment models and economically motivated scenarios, we define six scenarios (named SC1–SC6) with a varying number of access points for each type. Table 5.5 summarizes the different evaluation scenarios. For each scenario, the relative and absolute (in brackets) number of access points per type are shown. The scenarios are meant to provide an *exemplary* combination of percentages for the upgrade of access points, motivated by different underlying deployment models and constraints. For instance, by incentivizing private individuals to provide computing capabilities at their home routers, the number of devices is likely to increase, as reflected in SC4. Similarly, network operators and municipalities are likely to be subject to different cost constraints and economic goals. For instance, network operators might choose to upgrade their cell towers based on the average user density or demands at certain points in the city. Subsidies or regulatory action might be another way to influence the deployment. We reflect such variance by choosing 75 % (SC1 and SC2), 50 % (SC3 and SC4), and 25 % (SC5 and SC6) as the percentages for the cell towers. For the street lamps, 5 % (SC4), 10 % (SC2 and SC5), 25 % (SC1 and SC3), and 50 % (SC6)

[12] https://www.postgresql.org/ (accessed: 2019-11-06)

[13] https://postgis.net/ (accessed: 2019-11-06)

TABLE 5.4: PARAMETERS FOR DATA FILTERING

Speed threshold	$60\,\mathrm{km\,h^{-1}}$
Distance threshold	5 m
Time threshold	5 min
Minimum path extension	$2000\,\mathrm{m^2}$
Minimum data points per path	4

TABLE 5.5: EVALUATION SCENARIOS†

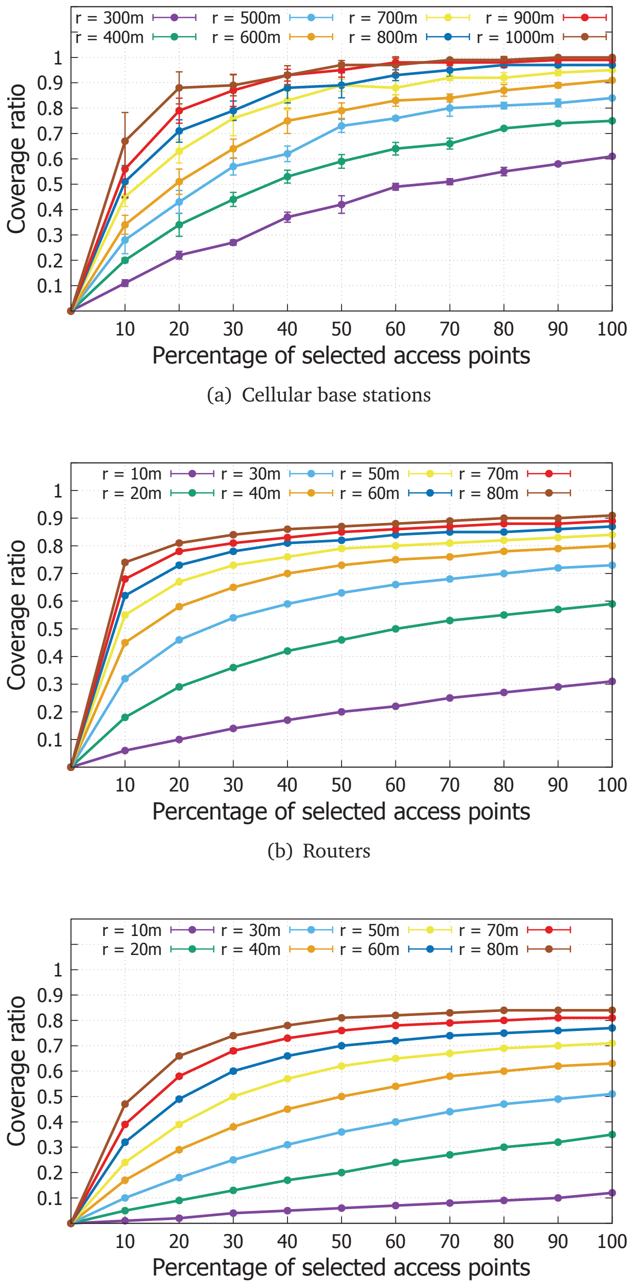
Scenario	Cellular base stations	Routers	Street lamps
SC1	75 % (68)	10 % (3224)	25 % (1433)
SC2	75 % (68)	25 % (8060)	10 % (573)
SC3	50 % (45)	25 % (8060)	25 % (1433)
SC4	50 % (45)	50 % (16 120)	5 % (286)
SC5	25 % (22)	25 % (8060)	10 % (573)
SC6	25 % (22)	10 % (3224)	50 % (2867)

of all access points are chosen. Given the lack of reference deployment models today, we argue that these scenarios allow for a first meaningful analysis of cloudlet coverage.

5.6.2 Spatial Coverage

Methodology. First, we investigate spatial coverage for the individual access point types without considering mobility traces of users. Figure 5.4 shows the results for cellular base stations (Figure 5.4(a)), routers (Figure 5.4(b)), and street lamps (Figure 5.4(c)). We assume a unit-disk model for the communication ranges—a common model for coverage in the domain of wireless sensor networks [Wan11; LC11]—and show the results for different realistic communication ranges for each type of cloudlet. For each number of access points (in steps of 10 percent), the corresponding access points are randomly chosen. Besides access points located inside the inner city boundary, we also include access points whose communication ranges span across that boundary. Each experiment is run five times and we plot the average coverage ratios. While the resulting plots also display the corresponding error bars, they are very small for routers and lamps. This is because their communication ranges are much smaller and they are more spatially distributed compared to cell towers. Therefore, overlaps in the communication ranges (that add to the variance of overall coverage when randomly selecting access points) are less likely. In contrast, cell towers have a bigger communication range. Thus, when randomly selecting access points, overlaps are more likely, and the gain in coverage might vary more significantly.

Results of the spatial coverage analysis. From the results, we can already observe the general trend that a rather small fraction of upgraded access points is



(a) Cellular base stations

(b) Routers

(c) Street lamps

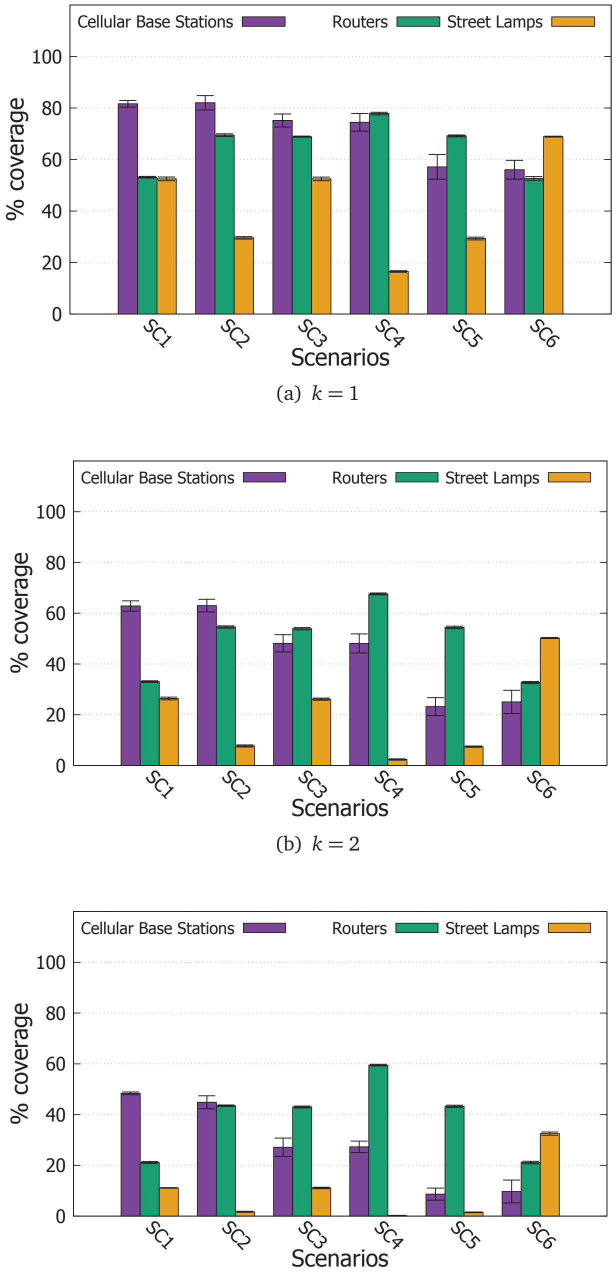
FIGURE 5.4: ANALYSIS OF SPATIAL COVERAGE†

sufficient to provide good spatial coverage. This is especially true for routers because of their sheer number. Assuming a rather conservative communication range of 40 m, already 20 percent of routers lead to almost 60 percent spatial coverage. For street lamps, the numbers are smaller because we have fewer lamp posts compared to routers. The same 20 percent result in about 30 percent coverage for street lamps.

The increase in coverage for routers and street lamps is slower from a certain point on because with increasing numbers, we get more spatial overlap in the communication range and, thus, less gain in overall coverage. In comparison, there are far fewer cell towers. However, cell towers have a much greater communication range today. Figure 5.4(a) shows three consequences of this. First, the coverage is lower for a small percentage of selected access points compared to routers and lamps. Second, adding more cell towers keeps increasing the overall coverage more significantly compared to routers and lamps, and third, intersections in the communication ranges lead to high values in the error bars for small percentages.

Spatial k-coverage. We now examine spatial k-coverage with the scenarios defined earlier. We again assume a unit-disk model for the communication ranges and select them randomly between the following ranges. For cellular base stations, the communication ranges are randomly chosen within a range of 300 to 1000 meters. Some works suggest an average communication range between 50 and 60 meters for WiFi routers [Mot+13; Bur+15]. However, in our urban scenario, this might vary greatly (e.g., due to obstacles or different building structures); therefore, for cloudlets on routers, we choose a range between 20 and 70 meters. In our analysis, we assume that in the near future, cloudlets on street lamps will be equipped with WiFi for communication. Because the wireless signal from the access points on street lamps is less obstructed compared to routers, we increase the range to 30–80 meters. Note that the lower value of the range is disproportionally higher because we assume the immediate surroundings of the lamp to be free of large obstructions (e.g., because in the deployment, the access points are mounted at a certain height). We use a uniform continuous distribution for all communication ranges and run each experiment 5 times. The results are shown in Figure 5.5. For each scenario, the bars represent the coverage per access point type.

In Figure 5.5(a) we consider 1-coverage, i.e., we assume an area to be covered if it is within the range of at least one access point. For 1-coverage, one can therefore think of this figure as a combination of the values shown in Figures 5.4(a)–5.4(c). As with our previous results, we see that a relatively small number of access points already provides good coverage. However, this significantly differs for the different types of access points. For instance, using only a small number of street lamps (as in scenario SC4) is clearly not viable in practice because of low coverage. Figures 5.5(b) and 5.5(c) show the results for $k = 2$ and $k = 3$, respectively. The biggest drop in coverage for increased values of k occurs with the street lamps, since they are spaced out more evenly and therefore, overlaps in the communication ranges occur less often. Since the communication ranges of cellular base stations and routers often overlap, we see that for $k = 2$ (Figure 5.5(b)) we still get good coverage. For $k = 3$ (Figure 5.5(c)), we can observe that routers lose the least coverage of all, possibly because of their generally higher degree of overlap. In future work, instead of randomly selecting access points, one might want to optimize the selection for a desired target value of k.



(a) $k = 1$

(b) $k = 2$

(c) $k = 3$

FIGURE 5.5: SCENARIO-BASED EVALUATION OF SPATIAL k-COVERAGE††

Summary of the spatial coverage analysis & outlook. From the analysis of spatial coverage, we can already see that deploying cloudlets is feasible for large-scale coverage within a city and that only a fraction of access points are required to achieve reasonable coverage. We expect the overall coverage ratio to be even better when taking into account the cover metrics that are based on mobility traces, i.e., point, path, and time coverage.

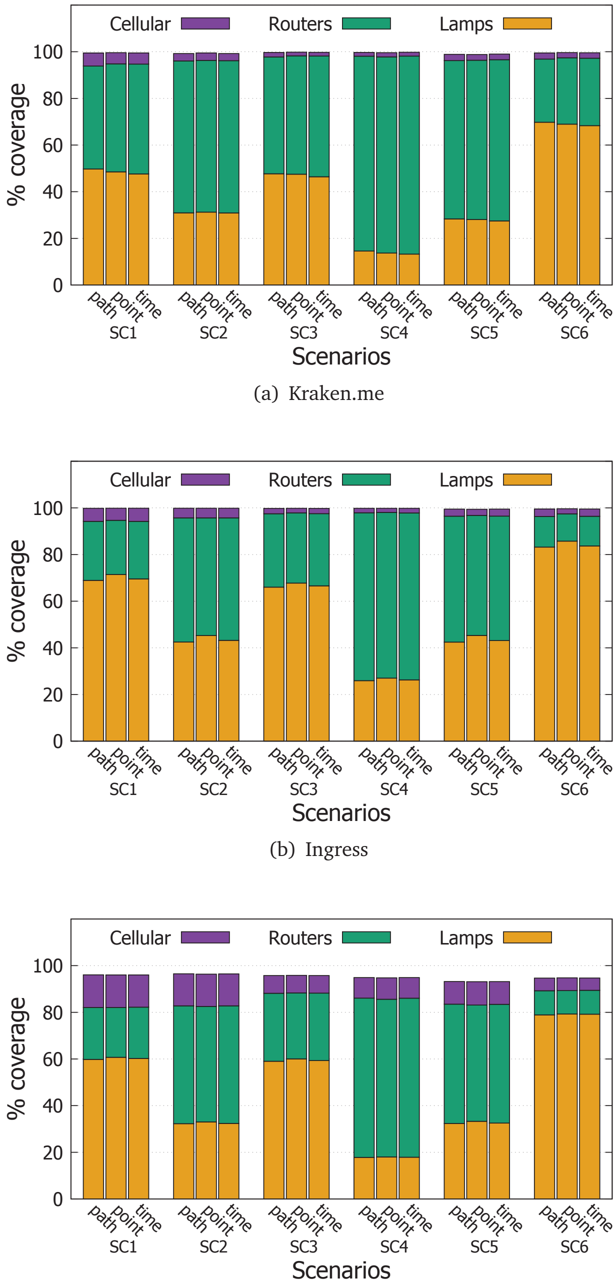
5.6.3 Point, Path, and Time Coverage

Methodology. While the results of Section 5.6.2 give an indication of how well an area is covered, one might argue that this does not necessarily represent the cloudlet coverage a mobile user experiences, since actual user locations are not uniformly distributed throughout the area. In addition, users are mobile and change their location frequently. Therefore, we now consider the mobility traces described in Section 5.4.2 and evaluate their point, path, and time coverage (as defined in Section 5.5). This analysis allows for more realistic insights regarding future offloading demands that could be served by cloudlets. For the communication ranges, we again use a uniform continuous distribution and a unit-disk model, randomly selecting within the following values according to the access point type: cellular base stations 300–1000 m, WiFi routers and street lamps 10–80 m.

Determining the combined coverage of access points. Figure 5.6 plots the results for the three datasets. In each of the plots, we evaluate the point, path, and time coverage per scenario. The individual bars are stacked to represent the combined coverage we obtain from multiple types of access points. The stacking represents the *additional* coverage we gain by adding the subsequent type of access point. We assume that as the first type of access point, street lamps will be chosen, since—giving the underlying business model of municipalities providing services to their citizens—they will incur the lowest costs for users. Furthermore, because most of our location traces are not inside buildings but outside, street lamps are likely to be closest and therefore the best-connected cloudlets for users in many cases. The next part of the bar represents how much coverage routers add to points, paths, or time spans not covered by those lamps. Since our model assumes cell towers to be the most expensive type, they are used last to fill the gap that cannot be covered by other types of access points. It is important to note that the height of the stack, i.e., the overall coverage would not change if we were to use another order of access points (say, if we would first assume coverage by routers and then add street lamps and cell towers).

Differences between coverage metrics. Surprisingly, the variance between the different types of coverage (i.e., path, point, and time coverage) is very small. While there are variances within a dataset with respect to the distance and difference in times between the data points, this averages out, i.e., we get nearly identical values for the different metrics of coverage. For datasets with a coarser temporal resolution, such as Ingress, we can, however, observe a lower time coverage compared to the other metrics.

Overall, this result shows that even if we have limited data (e.g., sparse location data for parts of a user's movement), this averages out over the whole dataset and hence, users will most likely also have a connection to a cloudlet for most of the



(a) Kraken.me

(b) Ingress

(c) CrowdSenSim

FIGURE 5.6: SCENARIO-BASED COVERAGE ANALYSIS OF PATH, POINT, AND TIME COVERAGE FOR THE MOBILITY TRACES†

time along their path. This provides an interesting insight for the planning of city-scale cloudlet infrastructures in the case where there is only limited data available to estimate the required demands. For instance, data protection laws might restrict the linking of entire user paths with their timestamps. Our analysis shows that regardless of what kind of data is available, each of the three coverage metrics can be used to estimate the resulting coverage for mobile users with offloading demands.

Comparison with spatial coverage. Looking at the overall coverage across the datasets, we see that the coverage is higher compared to the previous spatial coverage analysis because spatial coverage also includes areas that are less likely to be populated by people, e.g., in-between factory buildings or in more sparsely populated residential areas. This is validated by the fact that for the CrowdSenSim dataset (Figure 5.6(c)), which represents generated movement traces rather than real ones, the overall coverage is lower compared to Kraken.me (Figure 5.6(a)) and Ingress (Figure 5.6(b)). Even though the CrowdSenSim traces are generated on walkable paths, those are more likely to be in areas with limited coverage (e.g., because they are within less populated areas of the city). Recall that in all scenarios, only a certain percentage of each type of access point infrastructure is available. This explains why when using the CrowdSenSim traces in scenarios with a small percentage of cell towers (SC5 and SC6), we do not achieve full coverage compared to the other datasets. Overall, however, our analysis showed that with only a subset of available access point infrastructures, we are able to provide city-wide coverage.

Discussing the different access point types. From the results we could observe that, in general, when combining the different rather small percentages for the individual types, we get high overall coverage ratios. For instance, for the first scenario (SC1) of the Kraken dataset, selecting only 25 % of street lamps leads to almost 50 % of coverage for that type alone. Adding routers, which are present in much greater number, the coverage surpasses 90 %. This result holds true across all investigated datasets and coverage metrics.

Naturally, the percentage of lamps that is selected first has the highest impact on the distribution of access point types to their contribution to the coverage. For the Ingress data, the coverage values obtained only by the street lamps are slightly higher. We attribute this to the fact that different users congregate at distinct locations, i.e., where the game portals are located). Even for more fine-grained data (in the sense of capturing more data points along a user's path) like user locations from Kraken.me, we see that street lamps alone already achieve high coverage. As an example, scenarios SC1, SC3, and SC6 consistently lead to a coverage of well over 40 %. Routers are always able to fill up the coverage to over 90 %, except for the artificially generated traces because those traces are limited to streets, and therefore, fewer routers might be able to reach them. We can further observe that cell towers, which are likely to be most expensive and distant to the user, are still useful in filling coverage gaps and should still be considered in view of alternatives, such as local processing on the mobile device itself or cloud offloading.

5.7 Conclusion

In this chapter, we have examined the question of whether it is feasible to upgrade existing urban infrastructures to host cloudlets for a city-wide coverage. Specifically, we proposed placing cloudlets in an urban space using existing access point infrastructures, namely, cell towers, routers, and street lamps. We described the characteristics of these different types of infrastructure and gathered a large dataset of real-world data for access point locations and mobility traces. To analyze the coverage of urban cloudlets, we used these datasets and defined four different metrics for coverage. The results of this coverage analysis demonstrated that by using only a relatively small subset of access points present on infrastructure to host cloudlets, we are able to achieve a city-wide cloudlet coverage.

This is especially true for the coverage analysis of the mobility traces, where mobile users are within the communication range of a cloudlet-enabled access point most of the time. The results of this analysis enable different stakeholders (e.g., municipalities and network operators) to estimate the number of cloudlets required to achieve a certain degree of coverage and hence, can serve as a planning tool for future deployments. For existing deployments, it can identify areas that lack coverage (globally or for individual access point types or application traces). The developed method can serve as a basis for future analysis based on other data. For instance, instead of a unit-disk communication range, more sophisticated models for wireless signal propagation could be used to obtain a more realistic coverage estimation. As an example, for cell towers these models would come from measurements carried out by the network operators. Today, unfortunately, this kind of data is hard to obtain for public usage due to confidentiality reasons.

Our analysis demonstrated the feasibility of leveraging urban cloudlets. It is therefore the basis to examine the placement, i.e., to answer the question *where* exactly the cloudlets should be placed. In the following Chapter 6, we will address precisely this problem.

CHAPTER 6

Urban Cloudlet Placement

Chapter Outline

6.1 Introduction and Problem Statement

In the previous Chapter 5, we motivated the usage of existing urban infrastructures for the placement of cloudlets. We showed that placing cloudlets on those infrastructures can potentially achieve a high coverage. We demonstrated this with real-world data and four different coverage metrics. Furthermore, the methodology we presented can serve as a planning tool, e.g., to estimate the number of required access points for a certain percentage of coverage, or to identify uncovered areas.

However, the question remains of *where* to place those cloudlets, i.e., which infrastructure locations to equip with a cloudlet to serve the users' demands. Upgrading every access point with a cloudlet may not be feasible economically, and hence, decisions to equip a subset of all available access points have to be made. This problem is challenging for two reasons. First, the costs and the quality of service form a natural tradeoff. The quality of service can, for instance, be modeled as the amount of offloading demands that can be served by cloudlets. Second, as we have introduced in Section 5.2, the access point infrastructures we consider are highly heterogeneous, making the placement problem challenging. This chapter addresses this problem and proposes a strategy for the placement of heterogeneous urban cloudlets. For the placement decision of cloudlets in this chapter, we consider heterogeneity in three aspects:

(i) Costs | The different potential operators of a city cloudlet infrastructure and the heterogeneous hardware that can be used for cloudlets will lead to different cost structures and business models under which they operate. Our problem model (Section 6.3) accurately reflects the real-world economics of such deployments by capturing both fixed and variable costs.

(ii) Resources | Besides economic considerations, the physical environment of the access points dictate what type of cloudlet hardware can be placed at a given access point. This can range from small, embedded computers to server-grade hardware comparable to data centers.

(iii) Communication range | The different types of wireless access technologies have varying communication—and hence—coverage ranges. For instance, cell towers mounted on buildings have a higher coverage rate compared to WiFi routers.

After reviewing related work in Section 6.2, we define our system model for the placement problem in Section 6.3. Following these definitions, this chapter will address the cloudlet placement problem by proposing a placement strategy (Section 6.4) that is suitable for the described heterogeneous scenario. The proposed strategy runs in polynomial time and, hence, makes the problem tractable for large instances. Furthermore, it is able to trade the quality of service for costs, depending on different underlying economic considerations. Using the same real-world data introduced in Section 5.4, we will show how our approach outperforms two baseline strategies for placement (Section 6.5).

6.2 Related Work

Placing cloudlets on urban infrastructures faces the challenge of heterogeneity with regards to coverage, costs, and resources. In general, placement strategies can be optimized towards different metrics, e.g., the overall costs, or how much user demand can be served. In the scenario we outlined, placement strategies need to select *existing* locations for the placement instead of *defining* an optimal location. This is because we want to reuse existing infrastructure in order to minimize costs incurred by building new access points.

While there is abundant research on the placement of (virtualized) computing resources, both for homogeneous environments like data centers [MPZ10; PY10] and in the context of cloudlets and Edge Computing (see Section 8.2), the question of where to place cloudlets on available urban infrastructures has rarely been examined. Because urban infrastructures are highly heterogeneous, most existing placement strategies are unable to model the inherent tradeoffs (e.g., of costs versus coverage) in such environments. Consequently, they are not able to make sensible placement decisions.

Cloudlet placement. Three works [Xu+15; Xu+16; JCL17] study the placement of cloudlets in wireless metropolitan area networks (WMANs) and jointly propose solutions for the user-to-cloudlet allocation problem, but they do not consider the costs of cloudlets. Fan and Ansari [FA17] consider the costs for renting a hosting facility and cloudlet servers but their model does not capture variable costs per request—a crucial decision factor for the placement, given that the infrastructure is owned and operated by different stakeholders with varying business models (see Section 3.4). Similarly, Ren et al. [Ren+18] consider costs but assume that each cloudlet is able to meet all request demands within a region. We argue that this is not a realistic assumption, given the vast number of users and Edge Computing applications in an urban setting.

Xu et al. [Xu+15; Xu+16] present a greedy heuristic to minimize the average access delay of mobile users to a cloudlet. Ma et al. [Ma+17] operate on a similar model but use Particle Swarm Optimize (PSO) as a metaheuristic to outperform greedy approaches. Jia et al. [JCL17] devise two algorithms to minimize the response time: *Heaviest-AP First (HAF)* and *Density-Based Clustering (DBC)*. The former places cloudlets on the access points where user workloads are the heaviest, while the latter places cloudlets according to user-dense regions.

The authors in [SBD18] analyze a large dataset of cell tower locations in the US. Without considering the costs or computing resources, they investigate the distance reduction to data centers when cell towers of a certain category—classified according to the estimated residential population—are upgraded with micro data centers.

Yao et al. [Yao+17] investigate the cost-aware deployment of cloudlets that are heterogeneous with respect to costs and resource capacities. They adopt a greedy strategy that iteratively chooses cloudlets with the minimum unit cost of resources. Compared to our model, they make assumptions that we argue are not realistic, e.g., that there is no spatial overlap in the deployment of cloudlets and that the entire area is covered by access points. Even though the authors consider heterogeneous cloudlets, they are not linked to real-world infrastructure. In contrast, we consider three different types of infrastructure, each with specific characteristics.

Usage of WiFi or cellular access points. Caselli et al. [CPS15] focus on the planning of a cloudlet network that consists of cellular base stations only. Bulut et al. [BS13; BS16] have studied the deployment of WiFi access points; however, they consider only the access dimension and not the actual computations. This means that in their model, there is no capacity constraint for the placement of an access point. Furthermore, the authors do not model the cost of the deployment and assume that access points can be freely placed—the latter assumption can also be found in [Yin+17]. Mohan et al. [Moh+18] follow a hybrid approach where existing edge cloudlets are considered and new ones are freely placed. While the authors consider both WiFi and cellular access, they fail to model resource heterogeneity for different types of cloudlets. In contrast to that, we assume that we cannot influence the placement of the access points but instead have to choose a subset of the existing ones while taking into account the costs of cloudlet placement. In [BS13], the authors present a greedy algorithm for the access point placement that maximizes the offloading ratio, i.e., how much overall offloading can be achieved. This metric is extended to consider the satisfaction of individual users in [BS16]. Besides placing access points in areas with high overall demand, the presented placement strategy tries to enable offloading for each user equally.

Choosing the best access point. The work of Yang et al. [Yan+16] presents a method to rank access points in order to determine on which access points cloudlets should be placed. Their ranking function only considers the network features of the access point and ignores the cost dimension. Furthermore, the network model insufficiently reflects the characteristics of Edge Computing. For instance, the authors consider a metric of *closeness centrality*, defined as the average distance from an access point to all other access points. Zhao et al. [Zha+18e] also consider this metric besides others. Given that ideally in edge deployments, the flow of data is limited to the distance from the end device to the closest cloudlet and there is no dependency on other computing resources, we believe this metric does not accurately represent a placement utility in a typical Edge Computing scenario.

Summary. In conclusion, we could identify two major drawbacks of existing works in the domain of cloudlet placement: (i) failing to capture the relevant dimensions of heterogeneity for urban cloudlets (i.e., fixed and variable costs, coverage, and available computing resources), and (ii) making assumptions that are unrealistic for a real-world deployment (e.g., being able to freely place access points). Furthermore, no previous works have used extensive real-world data to evaluate their findings. In contrast, we jointly use location data of urban infrastructure and user traces captured from mobile applications (see Section 5.4) for the evaluation of our placement algorithm.

6.3 System Model

6.3.1 Basic Definitions

We consider a set $AP = \{ap_1, \dots, ap_n\}$ of n access points located in a 2-dimensional plane. Each access point $ap \in AP$ is of one type $t_i = Type(ap), t_i \in \{t_1, \dots, t_n\}$ and has a unit-disk communication range of radius r_{ap}. Following our dataset of urban cloudlets (see Section 5.4.1), we use cellular base stations, WiFi routers, and smart street lamps as access point types in this chapter. If an access point is chosen to be upgraded to host a cloudlet, it can provide a certain amount of resources R_{ap}, which for instance, can be modeled as the available CPU cycles of the cloudlet hardware. Our model reflects both fixed expenses and variable operational costs. This is a common abstraction found in economics (often termed CAPEX[1] and OPEX[2]). In detail, we consider the following two costs:

(i) FIXED COSTS | A fixed cost of $CFix_{ap}$ occurs when an access point is upgraded. This could either be the cost of upgrading hardware or fixed costs for running the cloudlet for a certain amount of time, e.g., the costs for energy.

(ii) VARIABLE COSTS | Variable costs of $CVar_{ap}$ occur per unit of computing resources used on the access point's cloudlet.

We introduce a binary decision variable $x_{ap} \in \{0, 1\}$ to model the placement of cloudlets on access points. $x_{ap} = 1$ if a cloudlet is placed on access point $ap \in AP$, 0 otherwise. From the mobility traces, we have m user locations, denoted as $U = \{u_1, \dots, u_m\}$. Each user requests a workload w_u. We further define $d(ap, u)$

[1] capital expenditures
[2] operational expenditures

as the Euclidean distance between an access point ap and a user location u. We characterize the association of a user to a cloudlet-enabled access point by $y_{u,ap} \in \{0,1\}$. If user u offloads the computations to a cloudlet present at access point ap, $y_{u,ap} = 1$, otherwise $y_{u,ap} = 0$.

A placement p is therefore defined as the assignment of the variables x_{ap} and $y_{u,ap}$. We denote all possible placements with $\mathbb{P}$. Our placement algorithm determines a placement for a fixed number of K cloudlets that are given as input, i.e.,

$$\sum_{ap \in AP} x_{ap} = K, K \in \mathbb{N}. \tag{6.1}$$

Placements are subject to a number of constraints. Obviously, users can only make use of a cloudlet at an access point if they are within its communication range and the access point has been equipped with a cloudlet, hence,

$$d(ap,u) \leq r_{ap} \forall u \in U, \forall ap \in AP : y_{u,ap} = 1 \tag{6.2}$$

and

$$x_{ap} \geq y_{u,ap} \forall u \in U, \forall ap \in AP. \tag{6.3}$$

We further assume that user demands cannot be fragmented. We make this assumption because fragmenting demands may result in other overhead, such as synchronization between the cloudlets, if parts of the same application are offloaded to more than one cloudlet. Hence, as a simplification, all workload demand from one user is offloaded to exactly one cloudlet and cannot be divided:

$$\sum_{ap \in AP} y_{u,ap} = 1, \forall u \in U \tag{6.4}$$

Placement decisions also need to consider the resource constraints on the cloudlets. Because user-to-cloudlet assignments should not overload the cloudlet, we have

$$\sum_{u \in U} y_{u,ap} \cdot w_u \leq R_{ap}, \forall ap \in AP. \tag{6.5}$$

To evaluate how good a placement decision is, we take into account two factors: the costs and the overall quality of service. Costs include the fixed cost for deploying a cloudlet as well as the variable cost for each unit of resources that is offloaded. Hence, the total costs of a placement can be formulated as

$$C(p) = \underbrace{\sum_{ap \in AP} CFix_{ap} \cdot x_{ap}}_{Fixed\ Costs} + \underbrace{\sum_{ap \in AP} \sum_{u \in U} y_{u,ap} \cdot CVar_{ap} \cdot w_u\,.}_{Variable\ Costs} \tag{6.6}$$

We model the quality of service as the ratio of how much user demand can be offloaded to the cloudlets, i.e.,

$$Q(p) = \frac{\sum_{u \in U} \sum_{ap \in AP} y_{u,ap}}{m}. \tag{6.7}$$

Compared with our previously introduced definitions of coverage (see Section 5.5), this is a variant of point coverage. However, for a point to be covered, in addition

Table 6.1: Notation of the placement problem†

AP	Set of access points
n	Total number of access points
R_{ap}	Available resources after upgrading the access point
r_{ap}	Radius of the unit-disk communication range
$CFix_{ap}$	Fixed cost for deploying and/or operating a cloudlet at the access point
$CVar_{ap}$	Variable cost for using one unit of resources
x_{ap}	Binary decision variable to indicate cloudlet deployment
U	Set of user locations
m	Total number of user locations
w_u	Workload requested at user location
$d(ap,u)$	Euclidean distance between access point and user
$y_{u,ap}$	Binary decision variable for user-to-cloudlet assignment
$C(p)$	Cost of a placement
$Q(p)$	Quality of service of a placement

to being located within the connectivity range of a cloudlet, the computational demands of the user at that point must be met, i.e., there must be a cloudlet with enough (remaining) computing resources in range. Given these definitions, the overall utility of a placement is defined as

$$Utility(p) = \alpha \cdot \frac{\max(C(\mathbb{P})) - C(p)}{\max(C(\mathbb{P})) - \min(C(\mathbb{P}))} + (1-\alpha) \cdot Q(p), \tag{6.8}$$

where $\alpha \in [0,1]$ is a weighting factor that tunes the impact of the two metrics $C(p)$ and $Q(p)$ on the overall utility of the placement. Practically, this allows to model different deployment scenarios, where either costs or the quality of service might be more important. As an example, for a value of $\alpha = 0.2$, the costs would account for 20 % and the quality of service for 80 % to the overall utility. Setting α to 0.5 would give equal weight to both factors. $\max(C(\mathbb{P}))$ and $\min(C(\mathbb{P}))$ denote the maximum and minimum possible costs of a placement. Note that we negate the cost factor in order to represent lower costs by a higher utility value. Table 6.1 summarizes the notation of our model.

6.3.2 Problem Definition

Given the above definitions, we define our cloudlet placement problem (CPP) as follows: Place K ($K \in \mathbb{N}$) cloudlets on access points according to the following optimization problem:

$$\begin{array}{ll} \max & Utility(p) \\ \text{s.t.} & (6.1), (6.2), (6.3), (6.4), (6.5) \\ & x_{ap} \in \{0,1\}, \\ & y_{u,ap} \in \{0,1\},\ \forall ap \in AP,\ \forall u \in U. \end{array}$$

Our goal is therefore to maximize the offloading ratio, i.e., the number of users that will be able to offload computations to cloudlets while making cost-aware placement decisions for cloudlets on the access points. Furthermore, we need to be able to compute placements for large problem instances. However, doing so is not practically feasible if we want to find the optimal solution for the CPP. We can model the CPP as a combination of two well-known problems: The *Metric Facility Location* and *k-Median* problem. The metric facility location problem describes the problem of finding a minimum-cost solution for opening facilities that are connected to cities. Costs occur for opening a facility and for the connection of a facility to a city. The k-median problem is a variation in which there are no costs for opening a facility and the number of opened facilities is upper-bounded by k. Jain and Vazirani [JV01] have shown that these problems are NP-hard. By setting cities to be users and facilities the cloudlets that need to be placed, we can construct an equivalent problem. The costs for opening a facility translate to the fixed costs for placing a cloudlet on an access point, while the costs for connecting a facility to a city can be mapped to the variable costs that have to be paid for processing user demands at the cloudlet. Hence, we can conclude that the CPP is also NP-hard. We will show in Section 6.4.1 that nonetheless, our proposed approach finds a solution in polynomial time.

6.4 Placement Strategy

To make the cloudlet placement problem more tractable, we propose GSCORE (Grid-Score), a cloudlet placement algorithm described in this section. The main idea of GSCORE is to perform cloudlet placement locally instead of on a global scale. To do so, we consider placement decisions on the level of grid cells, which divide the entire area. Grid cells may contain several access points on which cloudlets can be placed. Depending on the size of the grid cell, the communication range of an access point might span over multiple grid cells. Furthermore, to reduce complexity, we consider only *aggregated* demands in each grid cell, i.e., the sum of all user demands.

We divide the area to be covered by cloudlets into grid cells $G = \{g_1, \dots, g_j\}$ with uniform edge length gs. Based on the user locations and the request size of each user, we can then compute the total size of the requests per grid $w_g = \sum w_u$ for every user u located in that grid cell. Figure 6.1 shows an example of an area in the city that is divided into 9 grid cells $g_1, \dots, g_9$ ($|G| = 9$). It illustrates workloads $w_1, \dots, w_4$, represented by data points of user locations in that grid cell. In real-world deployments, the request sizes of the grids might be determined by measurements from network providers that are able to estimate the number of users and the offloading traffic they generate. Our algorithm operates solely on the knowledge of the individual grid cells. For each grid cell, a local decision is made to place a certain number of cloudlets on the available access points in that grid cell. First, we make a decision on where to place cloudlets and later assign the individual user requests to the cloudlets to evaluate the system utility as defined earlier.

(i) CLOUDLET PLACEMENT | The pseudocode of GSCORE is shown in Algorithm 1. Its main loop iterates over the grid cells until the desired number K of cloudlets have been placed (lines 1–27). The cells are traversed in decreasing order of request sizes, i.e., we begin with the cells that have the highest request sizes w_g (line 2). Next, for each of the access points located in that cell, a score is computed (lines 4–12). The score reflects the tradeoff between cost considerations and quality of service. A cost-to-resource ratio cr (line 5) is computed and a cost factor $factor_{cr}$ (line 8) normalizes the costs-to-resource ratio, taking into account the maximum and minimum cr values of

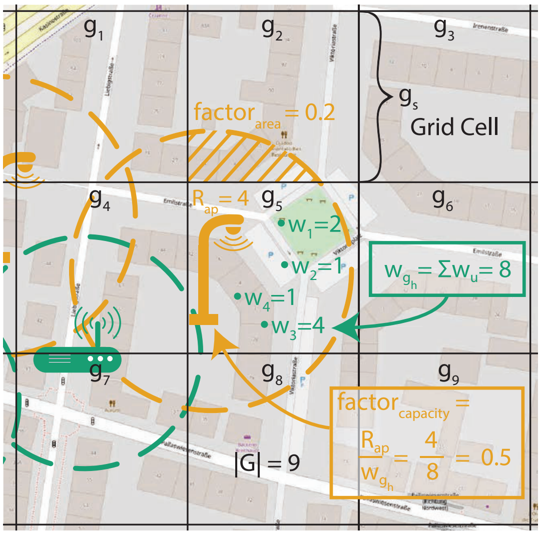


FIGURE 6.1: GRID MODEL FOR THE CLOUDLET PLACEMENT PROBLEM

all access points. Access points with higher resources at the same costs will therefore be ranked higher. For this cost factor, we assume an upper bound in the sense that each access point's capacity will be fully utilized.

Each access point covers a fraction of the grid cell's area (line 6) and is able to serve a fraction of this cell's workload demands (line 7). Both of these factors can have a maximum value of 1, in case the entire area is covered and all workloads can be served. The factor for the quality of service $factor_{QoS}$ (line 9) combines these two factors and therefore, reflects how much of the grid area is covered by an access point's communication range and what ratio of the grid's request demands can be satisfied by that access point. Note that we again only consider these factors on a grid cell level, i.e., a router with a larger communication range that covers an entire cell might have the same value for $factor_{area}$ as a cell tower, even though the latter in reality spans over multiple grid cells. Similarly, at this point, we disregard how many users actually are in the range of the access point (as they could be within the grid cell but not within the access point's communication range), and hence, how much workload could be served. We follow this approach because iterating over every individual data point would greatly increase the complexity of the placement decision. By selecting appropriate grid sizes in the evaluation, we will show that this approach is a reasonable approximation. Both factors can be weighted with a parameter $\alpha \in [0,1]$, in order to be more sensitive towards either costs or quality of service (line 10).

Figure 6.1 shows an illustrative example of how the capacity factor and area factor are computed in our grid model. In the given example, the street lamp is located in g_5. In that grid cell, it is able to serve half of the total workload requested in that cell. Its communication range is not limited to g_5, but spans across other cells. For example, its value for $factor_{area}$ is 0.2 in grid cell g_2.

From our raw user data, we could observe that the number of users per grid—and hence the generated request sizes—are not uniformly distributed. In-

Algorithm 1 GSCORE [†]

```
1: while $\sum_{i=0}^{n} x_i < K$ do
2:     $g_h \leftarrow G.getHighestRequestSize()$
3:     $S = \emptyset$
4:     for $ap \in AP$ located in $g_h$ do
5:         $cr \leftarrow \frac{CFix_{ap} + CVar_{ap} \cdot R_{ap}}{R_{ap}}$
6:         $factor_{area} \leftarrow \frac{|A(ap) \cap A(g_h)|}{|A(g_h)|}$
7:         $factor_{capacity} \leftarrow \frac{R_{ap}}{w_{g_h}}$
8:         $factor_{cr} \leftarrow \frac{max(cr) - cr}{max(cr) - min(cr)}$
9:         $factor_{QoS} \leftarrow \frac{factor_{area} + factor_{capacity}}{2}$
10:        $score_{ap} \leftarrow \alpha \cdot factor_{cr} + (1 - \alpha) \cdot factor_{QoS}$
11:        $S \leftarrow S \cup \{score_{ap}\}$
12:    end for
13:    $n \leftarrow \lfloor \ln(\frac{w_{g_h}}{\bar{w_g}}) + \ln(g_s) + \frac{K}{|G|} \rfloor$
14:    $placedCap \leftarrow 0$
15:    for $k \in [0, numToPlace]$ do
16:        $score_{ap} \leftarrow max(S)$
17:        $x_{ap} = 1$
18:        $placedCap \leftarrow placedCap + R_{ap}$
19:        $S \leftarrow S \setminus \{score_{ap}\}$
20:        $AP \leftarrow AP \setminus \{ap\}$
21:    end for
22:    if $(w_{g_h} - placedCap) > 2 \cdot \bar{w_g}$ then
23:        $w_{g_h} \leftarrow w_{g_h} - (placedCap \cdot \ln(w_{g_h}))$
24:    else
25:        $w_{g_h} \leftarrow w_{g_h} - placedCap$
26:    end if
27: end while
```

stead, we see few grid cells with substantially higher request sizes than the average. This will result in many access points being placed in those cells, even if they are not enough to satisfy the total user demands of that grid. At the same time, this reduces the number of (potentially more cost-effective) access points that could be placed for satisfying a larger offloading ratio in other grids. To mitigate this behavior, we compute the number of cloudlets to be placed in a grid cell, as shown in line 13 by the variable n. This formula normalizes the impact of cells with exceptionally high request sizes by taking $\ln(\frac{w_{g_h}}{\bar{w_g}})$. This normalization factor was determined empirically, given the distribution of requests across the grid cells. Note that this factor can be adapted to other datasets with different request patterns. However, we argue that in practice, it is in general realistic to assume non-uniform request sizes across grid cells, due to, e.g., users congregating at popular public places. In addition, we also factor in the size of the grid cell (in the sense that we allow more cloudlets to be placed in larger grids) and the ratio of K to the number of grid cells. According to this function, the corresponding number of cloudlets with the highest scores will be added to the grid cell (lines 15–21).

After having placed the corresponding number of cloudlets in a grid cell, its workload demand is adjusted in the following way: We assume each cloudlet will be used to full capacity. In addition, we again take into account the characteristics of the request size distribution to ensure that grid cells containing a smaller workload will also be iterated over. Hence, if the workload of a grid remains larger than two times the average workload (line 22), we adjust the new workload request estimation of the grid by multiplying the placed capacity of the cloudlets with the logarithm of the original request demand (line 23). Similar to the normalization factor for the number of access points to choose in line 13, we determined this value empirically.

(ii) User-to-cloudlet assignment | To compute the utility value (see Equation (6.8)), we now assign the requests of individual users with the following strategy: since the fixed costs have already been determined by the placement strategy, for each request, we choose the cloudlet with the lowest variable costs per resource unit that is within the range of the user. Note that for future work, other more sophisticated strategies could be used (see Section 6.6).

6.4.1 Complexity Considerations

In Section 6.3.2, we stated that the CPP is NP-hard. Contrary to that, our proposed placement strategy runs in polynomial time. The outer while-loop (lines 1–27) runs at most K-times. Within each iteration of the loop, we have to traverse the entire set of (remaining) access points to compute the score for the placement candidates (lines 4–12), giving a total (worst case) complexity of $O(|AP|)$. In addition, the second inner loop (lines 15–21) runs n-times. All other operations within the loops run in constant time $O(1)$. This also applies to the selection condition in line 4 that includes only access points located in the grid g_h. Spatial indices in the database can be used to retrieve access points that are within a geographic region in linear time [Ngu09]. Hence, the total time complexity of GSCORE is given as $O(K \cdot (|AP| + n))$.

6.5 Evaluation

6.5.1 Setup

We built a simulation tool in the Ruby programming language to evaluate our placement strategy. We use the filtered data for access point locations and user locations as described in Section 5.4. We use the values listed in Table 6.2 as our experimental settings for the modeling of access point attributes. The values reflect the heterogeneity of our access point infrastructure and different deployment characteristics. It is important to note that even within one type of access point, we consider the values for the communication range, resources, and costs to be variable. The only exception are the costs for using street lamps, which we assume to be fixed because they are operated by a single stakeholder, the municipality (see Table 5.1). Section 5.2 detailed how the different types of cloudlets are able to host varying computing resources according to the physical space and deployment models. In

TABLE 6.2: EVALUATION PARAMETERS[†]

	Cellular base stations	Routers	Street lamps
Communication range (m)	random (300,1000)	random (10,70)	random (20,80)
Resources	random (2000,5000)	random(5,100)	random(5,50)
Fixed cost	random (1000, 10 000)	random(1,100)	100
Variable cost	random(5,10)	random(1,5)	1

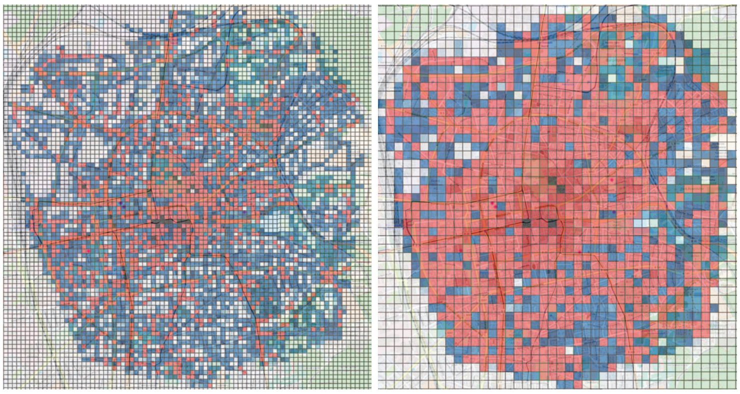
Section 5.6.2, we have furthermore investigated sensible values for the communication range of different types of cloudlets. From these considerations we derive the values for the communication range and resources listed in Table 6.2.

To model the workload that users want to offload, we take the data points from all three datasets presented in Section 5.4.2. Even though they were captured over a period of time, for the evaluation, we assume they jointly represent demand spots of mobile users throughout the city at a single point in time. This simplification allows us to generate sufficient demands, since we only have a limited number of data points. Each data point is assigned a requested workload of 1 or 2 units. We conduct our experiments with two different grid cell sizes of 50 meters and 100 meters for their edge lengths. Heatmaps of these two setups that visualize the total number of requests per grids are shown in Figures 6.2(a) and 6.2(b). Darker red squares represent grid cells with high demands, while blue ones are areas with low demands. As the access point locations, we use the dataset presented in Section 5.4.1.

We compare our placement approach with two alternative strategies for cloudlet placement, one that randomly selects cloudlet locations and one that solely optimizes costs. Both of these strategies do not operate on grids, but make global decisions for the placement.

Random (RND): This approach randomly selects K access points where cloudlets are placed on. Obviously, the distribution of the K selected access points with respect to their type will follow the one of the dataset, meaning that we will have few expensive locations (i.e., cell towers), and a high number of routers and street lamps. They will however not necessarily be located in areas where the coverage has a high impact on the QoS, i.e., areas with a large number of users. Instead, cloudlets are likely to be spread evenly throughout the city.

Greedy Cost-Aware (GC): This strategy tries to minimize the overall costs by iteratively selecting the access points with the lowest overall costs, defined as the sum of fixed and variable costs, assuming the placed cloudlet will be used to full capacity. Similar to RND, this will disregard the geographic distribution of user workloads and might penalize choices that have higher costs but a good costs-to-resource ratio. We therefore expect this approach to perform worse than RND in terms of the delivered quality of service. However, for very cost-restricted deployment models, this will lead to the insight of how much offloading is possible.



(a) Grid cell size of 50m

(b) Grid cell size of 100m

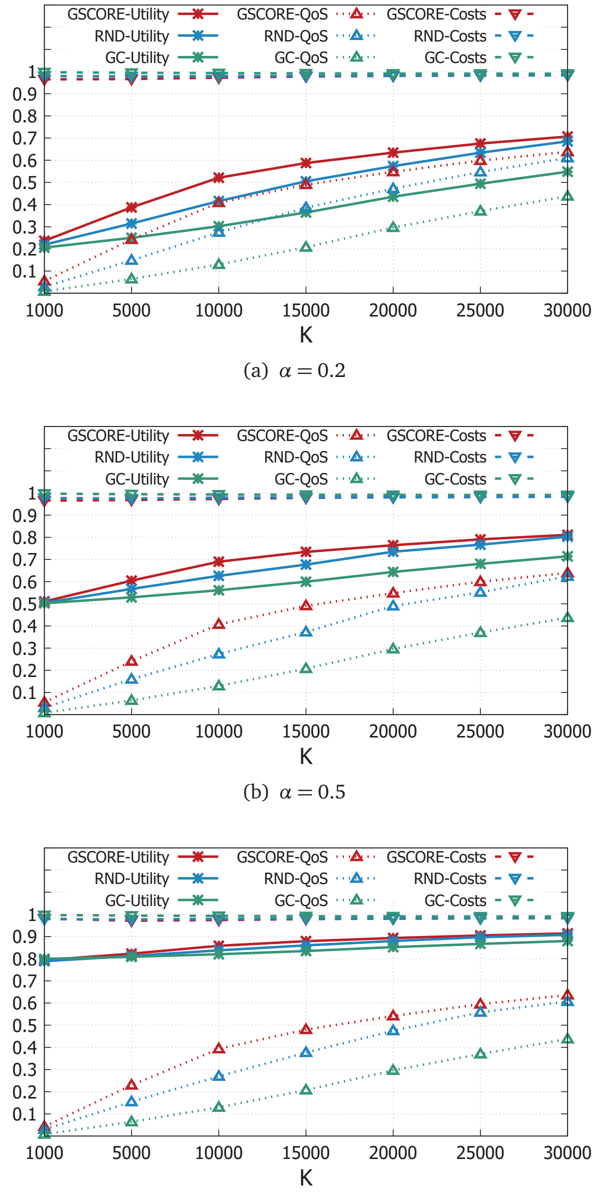
FIGURE 6.2: GRID CELL SIZES FOR EVALUATION[†]

6.5.2 Results

Figures 6.3 and 6.4 display the results of our placement strategy for grid cell sizes of 50 m and 100 m, respectively. For each grid cell size, we evaluate the placement strategies with values 0.2, 0.5, and 0.8 for α. Recall that $\alpha = 0.5$ equally weighs the factors for costs and quality of service in the overall utility, while lower values of α put more emphasis on the quality of service and vice-versa. For K, we use values from 1000 to 30 000. Recall that the total number of access points is about 37 000, hence, our evaluation aims to cover a reasonable range from upgrading very few up to nearly all access points. Besides the overall utility, the plots include the two components of the utility function, i.e., the values for the costs and QoS. As defined in Equation (6.8), higher values denote lower costs and better QoS.

Overall, we see that our proposed GSCORE algorithm surpasses RND and GC in each evaluated scenario for both the overall utility and the QoS part of the utility function. Even though barely visible in the graphs, GSCORE leads to a very small increase in the cost factor (in most cases around 1–2%) compared to the other strategies. However, the gain in terms of QoS when using GSCORE to place cloudlets is much higher. Take as an example the results for $K = 10\,000$ and $\alpha = 0.2$. Irrespective of the grid size, GSCORE achieves a QoS value that is three times higher compared to GC. Therefore, the essential benefit that GSCORE provides is that it trades a small fraction of cost increase for a much greater increase in the quality of service. Consequently, for smaller values of α, the gain in the overall utility when using GSCORE is higher. Since it takes into account the values of α for the scoring, GSCORE can be tuned to adapt to different deployment and business models for cloudlets.

Regarding the different grid cell sizes, we observe only a small difference between sizes of 50 m and 100 m. For 50 m, GSCORE gains a little in the overall utility. This is because smaller grid cells allow for a more accurate placement decision, since an access point located within a grid cell is likely to cover more of the area of that cell. Compared to larger cells, this leads to less cases where an access point is chosen that only covers a fraction of the cell's area, and, consequently, is able to serve fewer users. However, smaller grid sizes result in iterating over more grids,

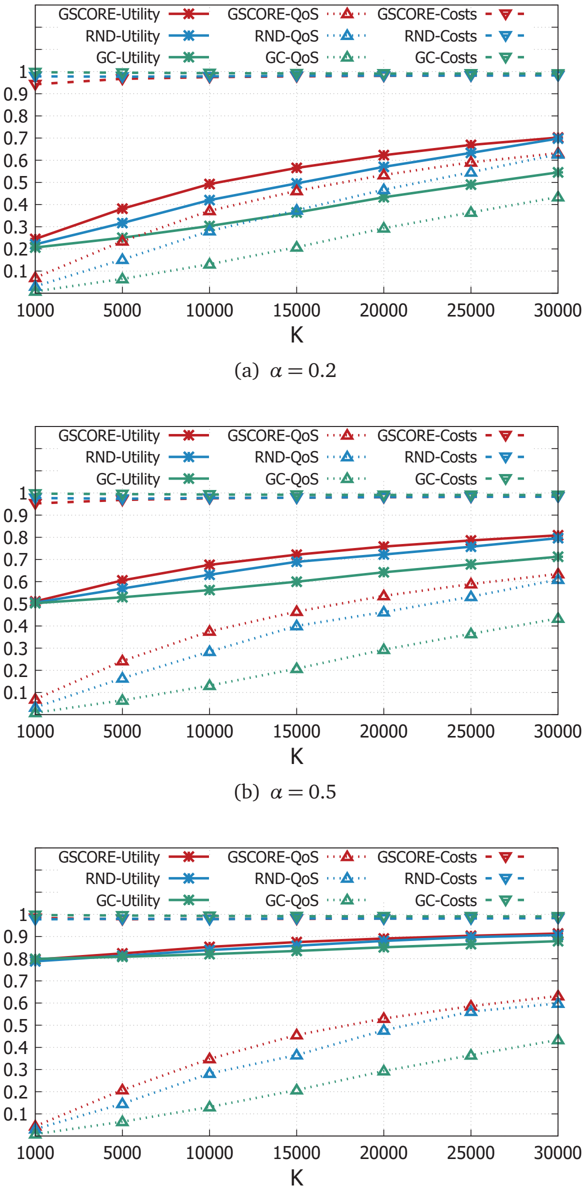


(a) $\alpha = 0.2$

(b) $\alpha = 0.5$

(c) $\alpha = 0.8$

FIGURE 6.3: PLACEMENT EVALUATION FOR A GRID CELL SIZE OF 50 M†



(a) $\alpha = 0.2$

(b) $\alpha = 0.5$

(c) $\alpha = 0.8$

FIGURE 6.4: PLACEMENT EVALUATION FOR A GRID CELL SIZE OF 100 M[†]

and hence, a greater computational overhead to make the placement decision. We leave the exploration of this tradeoff for future work and plan to further investigate how other grid cell sizes perform.

As we can also see, for larger values of K, the difference between GSCORE and RND becomes smaller because with increasing K, there naturally is a greater overlap in the subset of the chosen access points. This can be seen in the result plots if we look at the results for $K = 30\,000$ and compare this value with the total number of access points in the inner city boundary (37 648). The results for small values of K (e.g., $K = 1000$) behave similarly. In that case, there is less overlap but so few cloudlets selected such that not many user demands can be met, regardless of the placement strategy. GSCORE performs best for input sizes between $K = 5000$ and $K = 20\,000$ or in other words, between about 13 % and 53 % of all available access points. We therefore believe this placement strategy can be viable in practice, since in a real-world deployment, one would neither upgrade very few (because there would be no substantial gain for users) nor nearly all access points (because of practical limitations in terms of costs). Furthermore, we argue that GSCORE would perform even better compared to RND if the environment is more heterogeneous than in our evaluation setup, e.g., if cloudlets on street lamps are owned by different operators and therefore have different costs associated with them. Note that our notion of placing K cloudlets can easily be adapted to match other constraints, such as the total costs. Instead of K denoting the number of access points, K could for example model the maximum allowed fixed costs of the deployment.

6.5.3 Discussion

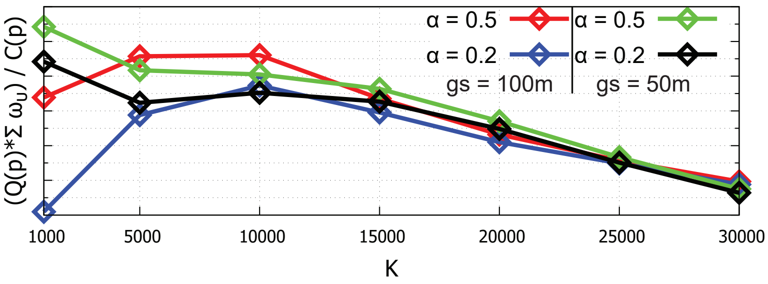


Figure 6.5: Choosing K by quality-to-cost ratio†

To further investigate practical insights on how to choose a suitable K, we analyze the quality-to-cost ratio for different values of K. Instead of using the normalized utility (see Equation (6.8)), we consider the absolute costs $C(p)$ and for the quality of service, we weight the absolute workload values by our QoS part of the utility. Hence, the ratio is given as

$$\frac{Q(p) \cdot \sum w_u}{C(p)} \tag{6.10}$$

The reason for considering the absolute costs in this analysis is to give insights on suitable values of K under real-world constraints, e.g., a fixed budget. Figure 6.5 plots the value for this ratio for different values of K. For this analysis, we use the placement decision that GSCORE outputs and the same parameters as described before. As α, we use 0.2 and 0.5. We omit $\alpha = 0.8$ because this shows a very clear trend towards very low values of K, meaning that except for $K = 1000$, the values of Equation (6.10) are very low. This is because $\alpha = 0.8$ weighs the cost factor with 80 % in the total utility.

From the plot, we can observe that the size of the grid cells only has a substantial influence for low values of K. At $K = 1000$, the quality-to-cost ratio is lower for a cell size of 100 m compared to a size of 50 m. This is because, for these low numbers, the placement algorithm placed almost double the number of cloudlets on cell towers for the larger grid cell size of 100 m. The reason behind such placement decisions is that larger grid cells will lead to extremely high computation demands in some cells (e.g., those that span an entire public square in a city). If a high-capacity cloudlet (i.e., at a cell tower) is located inside that grid cell, GSCORE will assign this location a high $factor_{capacity}$ score, and, therefore, it is likely to be chosen as a location. In comparison, a smaller size of a grid cell will make it less likely that a cell tower is located precisely *within* a grid cell with a high demand. For larger values of K, i.e., when we choose more cloudlets, this effect is less striking in the results. If we omit $K = 1000$, we can see a sweet spot for the value of K at about 10 000 across all grid cell sizes. It is worth noticing that these are also the ranges of K where our proposed algorithm performs best (in terms of surpassing RND and GC as depicted in Figures 6.3 and 6.4). For a very large number of cloudlets ($K \geq 15\,000$), the quality-to-cost ratio decreases and is the same across all grid cell sizes and values of α. This is because for such large numbers of cloudlets, the additional costs incurred surpass the gain in quality of service.

TABLE 6.3: NUMBER OF CLOUDLETS PLACED FOR A GRID CELL SIZE OF 50 M

	RND		GC		GSCORE	
	$k = 10\,000$	$k = 15\,000$	$k = 10\,000$	$k = 15\,000$	$k = 10\,000$	$k = 15\,000$
Cellular	26	34	0	0	33	36
Routers	8536	12 737	8473	11 075	7713	11 825
Lamps	1438	2229	1527	3925	2254	3139

TABLE 6.4: NUMBER OF CLOUDLETS PLACED FOR A GRID CELL SIZE OF 100 M

	RND		GC		GSCORE	
	$k = 10\,000$	$k = 15\,000$	$k = 10\,000$	$k = 15\,000$	$k = 10\,000$	$k = 15\,000$
Cellular	29	46	0	0	29	35
Routers	8447	12 674	8565	11 170	8152	12 428
Lamps	1524	2280	1435	3830	1819	2537

Lastly, we look at how the placement strategies differ with regards to the number of cloudlets placed on each type of access point. We investigate this for values of $K = 10\,000$ and $K = 15\,000$, the two K that lead to good quality-to-cost ratios for both grid cell sizes. Tables 6.3 and 6.4 show the numbers of placements for grid cell sizes of 50 m and 100 m, respectively. Obviously, for RND, the distribution of cloudlets on the three types follows exactly the distribution of available access points (compare Table 5.2). We see that GC always avoids placing cloudlets on cell towers, due to their high cost. Compared to RND, GSCORE places more cloudlets on cell towers, in order to increase the quality of service. Note that if we would drastically increase the user demands, this would increase the number of cloudlets placed cell towers, since cloudlets on routers and lamps would not be able to cope with the demands. Across both grid cell sizes, we further observe that for $K = 10\,000$, GSCORE places more cloudlets on lamps and less on routers, compared to GC. The reason for this is that for street lamps, we assume costs to be homogeneous, while for routers, those vary (see Table 6.2). Similar to the placement decisions on cell towers, GSCORE again trades a potential (small) increase in cost for greater improvement on the quality of service.

6.6 Conclusion and Future Work

In this chapter, we presented GSCORE, a placement algorithm to decide which urban access point infrastructures to equip with cloudlets. We showed that GSCORE is able to capture the heterogeneous characteristics of the urban cloudlet infrastructure and outperforms two baseline algorithms, RND and GC, while running in polynomial time. It does so by making decisions on grid cells of fixed sizes and is able to trade a small portion of cost for a substantially higher gain in the number of users that can offload computations. This tradeoff can furthermore be adjusted to capture different underlying business models and incentive mechanisms. We envision the following possible directions for future work:

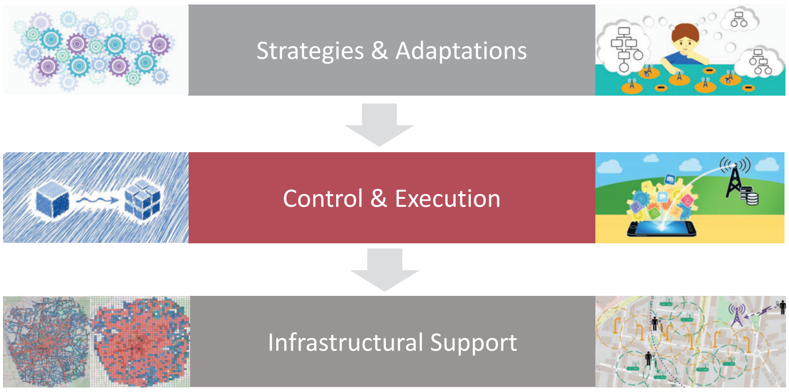
Other dimensions of heterogeneity | Besides cost, resources, and communication ranges, future work could consider other dimensions for the heterogeneity of access points, such as the available bandwidth or other network properties.

Assignment of users to cloudlets | In Section 5.5.1.a, we have shown that users might have connectivity to more than one cloudlet at a given location. In our placement strategy, we have assumed that user requests will be directed to the cheapest available cloudlet. However, other factors could be considered. For instance, users with high mobility could induce a frequent change of cloudlets, leading to a high overhead when migrating data or computations.

Dynamic grid cell sizes | Instead of uniform grid cell sizes, they could be varied, e.g., according to the workloads requested in the grid. This can serve the purpose to further optimize the placement strategy. For instance, neighboring grid cells with low demands could be merged to one, saving computation time for the placement decision, while at the same time not having a large impact on the overall quality of service.

Part III

Control & Execution



Part II examined the *physical infrastructure* for Edge Computing in an urban environment. On top of this infrastructure, we can now build the *software platform* for the execution of services at the edge. Such a platform manages the resources provided by the physical infrastructure, e.g., the cloudlets, and—through user-facing interfaces—receives requests for Edge Computing services and directs them to available resources.

In this part, we propose a control and execution framework for Edge Computing that is based on the concept of composable microservices. Chapter 7 will present this framework and details a novel approach to computation offloading by *onloading* parts of applications through a *microservice store*. The evaluation shows how this approach enables low-latency and energy-efficient execution of services at the edge.

CHAPTER 7

Edge Computing Framework[1]

Chapter Outline

7.1 Introduction

The preceding Part II of this thesis investigated the physical resources in an urban environment that are necessary to perform Edge Computing. Because of the multitude of users and applications in the context of Urban Edge Computing, computations in the context of Urban Edge Computing should not run directly on those resources without an intermediate control and execution layer. Besides enabling multi-tenancy of clients and applications, such a layer should be responsible for managing resources and the lifecycle of applications at the edge. Most importantly, this includes mechanisms for clients to perform computation offloading, i.e., to make parts of their applications available for remote execution. Furthermore, the control layer makes runtime decisions, e.g., where to place computations based on available resources and application requirements.

There are many previous works that propose frameworks for computation offloading, with varying granularity of the offloading units, e.g., the offloading of

[1] Parts of this chapter are verbatim copies from [Ged+19b]. Those text segments are printed in *gray color*. Tables and figures taken or adapted from this publication are marked with † in their caption.

methods [Cue+10], threads [Gor+12] or virtual machines [Shi+13]. Some aim at offloading to cloud infrastructures [BGG19; Chu+11; Shi+14], while others specifically target Edge Computing [LWB16; Mor+17; Jia+19]. Other works focus on the offloading decision itself [Che+16], or on security aspects [Bha+16].

Common to all of them is the tight coupling between mobile clients and the offloading infrastructure with regards to the offloading units. Mobility of those offloading units is realized by either (i) embedding the code and execution environments into virtual machines or containers that are transferred from the client to the surrogates that execute them, or (ii) pre-provisioning the offloading units on the surrogates. In the first case, this requires energy for the transmissions and these transfers add to the end-to-end latency, i.e., the time it takes from the service request to return the result to the user. For devices like smartphones, both of these factors are critical. Mobile users are often faced with unreliable, low-bandwidth mobile networks and limited battery life. Performing traditional offloading hence has a negative impact on these factors and thus affects the overall quality of experience. The second case—where offloadable units are pre-provisioned on the surrogate—is practically infeasible in an Edge Computing environment, where we are faced with limited resources on the surrogate and frequently changing demand patterns.

Many approaches that propose microservice-based execution environments fall short in their efficiency when multiple microservices and users are involved. More specifically, many lack mechanisms for the definition and deployment of *chained microservices*, i.e., a set of microservices that are executed subsequently. In addition, current approaches do not support the sharing of service instances between multiple users and applications and therefore cannot use the overall resources efficiently.

In this chapter, we propose the novel concept of *onloading* functionality to edge surrogates through a *microservice store*. The microservice store is a repository to which developers can submit their code in the form of containerized applications. Onloading fetches services from this repository and transfers them to agents that execute them. This stands in contrast to client-based *offloading*, where client devices transfer the services. Our system performs onloading on a microservice-based granularity. Microservices are an increasingly popular software development pattern with many advantages, e.g., higher agility and flexibility in the development of individual services. Developers of edge applications can leverage externally developed microservices from the microservice store as building blocks for their applications. At runtime, the applications request the instantiation of microservices and do not need to transfer code blocks and execution environments to the surrogate. Microservice instantiation can also be customized, e.g., by overriding the default lifetime of the service, and composed to form a chained execution of services. Microservices in the store can therefore be thought of as a blueprint. We define such a blueprint with the help of a common language for the orchestration of cloud services. Through the reuse of services for different applications, our approach also allows for a system-wide management of resources, e.g., by deciding which services are kept active, given request patterns from applications.

Orthogonal to existing frameworks for computation offloading, the concept of *Serverless Computing* has recently surfaced. While Serverless Computing partly removes the need for transferring code prior to its execution, it is mainly aimed at Cloud Computing applications and therefore lacks many aspects that are beneficial in an Edge Computing environment, such as the sharing of function instances or fine-grained control over the placement decision.

In summary, this chapter makes the following contributions:

- After reviewing related work in Section 7.2, we propose a programming model for Edge Computing based on the concept of microservices (Section 7.3). The model allows developers the definition and composition of edge-enabled applications. As an alternative to current offloading mechanisms, we propose a new mechanism for executing computations at the edge, coined *store-based microservice onloading*.
- We design the functional concept of a distributed Edge Computing framework that integrates the abovementioned concepts (Section 7.4) and leverages a customized repository for the provisioning of microservices, reducing expensive transfers from the client device to the surrogates.
- We realize our concepts in a prototype implementation called *flexEdge* (Section 7.5) and show the benefits w.r.t. end-to-end latency, energy savings on the client device, and efficiency of the service chaining (Section 7.6).

7.2 Related Work

Our contributions presented in this chapter aim at providing an efficient mechanism (with regards to end-to-end latency and energy consumption on the client) to carry out computations outside a client device. In addition, execution frameworks for Edge Computing should follow a modular structure that allows the reuse of (i) application components, and (ii) running instances of those components across applications. Such a reusable and modular structure is beneficial from the point of view of developers (as it shortens the development time of edge-enabled applications), and given the typically constrained resources at Edge Computing nodes. In light of these requirements, we review related work in the domains of computation offloading (Section 7.2.1), microservices (Section 7.2.2), and Serverless Computing (Section 7.2.3).

7.2.1 Computation Offloading

Computation offloading is the process of remotely executing an application or parts of this application [Kum+13]. This concept is sometimes also referred to as *cyber foraging* [Bal+02]. As detailed in Section 3.5.1, computation offloading is an important building block for Edge Computing, as it enables the carrying out of computations outside the end device.

Sharifi et al. [SKK12] review this general concept and summarize research challenges. Balan and Flinn [BF17] argue that challenges like server setup and maintenance are still not solved in current cyber foraging systems. We believe our approach at least partly frees developers from these burdens, as our Edge Computing platform is responsible for the instantiation and management of services. Others have focused on the simplicity for the developer to adapt applications for offloading. For example, Balan et al. [Bal+07] present a domain-specific language for the easy partitioning of applications.

According to Lewis et al. [Lew+14], three important questions need to be answered in computation offloading: what to offload, where to offload to, and when to offload. To these three well-established questions, we add a fourth by investigating *how to offload*, i.e., we propose an alternative to prevailing offloading mechanisms. Flores et al. [Flo+15] analyze the different components of an offloading system (e.g., code profilers and decision engines) and the practical limitations of current offloading approaches (e.g., with regards to the postulated energy benefit for mobile devices).

A variety of offloading frameworks have been proposed. Common to most of them is that code—and in some cases the entire execution environment—has to be transferred from the client device to the surrogate, e.g., as done in [KB10]. *Surrogate* is an umbrella term used for devices that execute code or applications on behalf of others [SKK12]. Other works, e.g., Wu et al. [Wu+17b] focus only on the efficiency on the surrogate side and not from the point of view of the client device. Furthermore, contrary to our approach, not many approaches offer the possibility for seamless execution of service chains, i.e., executing services subsequently without transmitting intermediate results to the client or to a control entity. In some cases [Cue+10; Chu+11] chaining is only implicitly enabled by the decisions of partitioning mechanisms. Some works focus on the speedup that offloading can provide [KYK12], while the main goal of others is to reduce the energy consumption [Cue+10; MN10], or the monetary cost in cloud offloading [Shi+14]. We can further distinguish offloading systems by the granularity in which applications or parts thereof are offloaded. Coarser-grained approaches offload entire virtual machines [Shi+13; SF05], while finer-grained approaches offload code parts [Kos+12; Cue+10], functions [Mor+17], or threads [Gor+12; Chu+11].

Cuervo et al. [Cue+10] present *MAUI*, an offloading framework for mobile phones that focuses on the energy benefit of offloading. Offloadable parts of an application are defined by code annotations. These annotations are static and made by the developer on a method-level granularity. Other works like *CloudAware* [OBL16] also require developer annotations for the offloading and operate on a method-level (e.g., *ThinkAir* by Kosta et al. [Kos+12]). We argue that annotations on such fine-grained units (e.g., single methods of applications) are cumbersome for the developer. In contrast, our framework *flexEdge* only requires the developer to request for a certain functionality provided by a microservice. Deploying applications in MAUI has a large overhead because two versions of the application need to be deployed, one locally and one on the surrogate. At runtime, the execution can then be switched between those two versions. This is done by an optimization engine that decides which annotated methods are offloaded. Compared to that, we use microservices that do not need to be present in the client version of the application. MAUI is also less flexible in the sense that it only supports the .NET common language runtime.

In contrast to partitioning via manual annotations, *CloneCloud* [Chu+11] automatically partitions the application. Hence, the offloading decision is made without developer involvement. A static analyzer identifies possible choices for code parts to be migrated. Then, a dynamic profiler constructs possible execution trees with associated costs. An optimization solver then makes the offloading decision at runtime. CloneCloud supports only Java applications on the *Dalvik* VM. *AIOLOS* [Ver+12b] also works exclusively on the Dalvik VM. Similar to MAUI, CloneCloud requires a complete clone VM on the surrogate. The application state has to be synchronized between the VMs, adding to the overhead of this solution. Similarly, the work of Gordon et al. [Gor+12] that uses distributed shared memory for offloading requires synchronization between endpoints. Besides only providing offloading functionality, *Thinkair* [Kos+12] also supports the parallelization of tasks and automatic scaling across several VMs.

Similar to our proposed approach, some other works [LWB16; Mor+17; Bha+16] also employ a repository for offloadable parts that are then transferred to surrogates. Paradrop [LWB16] is a platform for the dynamic orchestration of third-party services at the edge. Contrary to our work, the authors do however not consider the aspect of energy savings for the mobile device. *CloudPath* [Mor+17] is restricted to stateless functions. Both Paradrop and CloudPath do not allow the chaining of services. The work of Bhardwaj et al. [Bha+16] focuses on the security aspect of onloading.

Some previous works are restricted to a specific application domain, e.g., *Odessa* [Ra+11] builds on top of an existing stream processing framework. Mazza et al. [MTC17] present a concept for a cloud-based offloading mechanism for smart city applications that jointly manages computation and communication resources. However, the authors provide no implementation details and evaluation of the proposed concept.

To summarize this section, Table 7.1 compares the most relevant existing approaches for offloading. For each approach, we list the evaluation metrics they use, the granularity of offloading, which transfers or pre-provisionings are necessary, and whether service composition and instance reuse is supported. The bottom row of the table also shows how our approach compares to the existing ones.

7.2.2 Microservices

Microservices are a way to develop and deploy software as independent parts, in contrast to monolithic software [Fow14]. It is widely recognized that this offers many benefits regarding DevOps [BHJ16]; [KLT16]. The term *DevOps* summarizes different practices that use agile methods in order to achieve short development cycles and automated delivery of software [Ebe+16]. Dragoni et al. [Dra+17] provide a more in-depth introductory survey about the general concept of microservices.

Although the granularity of a microservice is not clearly defined [HAB17; HB16], microservices are typically characterized as small parts of an application with limited responsibilities, often restricted to performing a single task. Since those parts are developed and maintained independently, the concept of microservices allows for the reuse of components and supports multiple programming languages across the components of an application. Hence, developers can choose the most appropriate programming language and set of tools for the individual task. Furthermore, individual parts can be scaled more easily [Jam+18; Dra+18]. The concept of developing applications as microservices also brings new challenges and concerns [TLP17; STH18], e.g., with regards to an increased operational complexity and testing efforts.

Microservices have been used in practice for applications in Cloud Computing [Vil+15a; Vil+16a], IoT [SLM17; BGT16], and smart cities [KJP15]. Our proposed Edge Computing framework operates on a microservice-level granularity. The separation of multiple offloadable parts has advantages in the context of Edge Computing. For example, different application parts might have different resource requirements that cannot be met by all edge surrogates. To support different programming languages, we require our microservices to be shipped as containers. This concept of delivering microservices as containers had previously been suggested in [Sil16] and [Ala+18].

System	Evaluation metrics	Offloading granularity	Prior transfers before execution	Service composition	Instance reuse
MAUI [Cue+10]	Energy consumption, latency	Code (method)	Pre-deployed application clone	○ (implicitly by partitioning decision)	✗
CloneCloud [Chu+11]	Energy consumption, latency	Code (thread)	Pre-computed partition and application state	○ (implicitly by partitioning decision)	✗
ThinkAir [Kos+12]	Energy consumption, latency	Code (method)	Application transferred from client	✗	✗
Paradrop [LWB16]	Deployment time, CPU load	Microservices	From store to surrogate	✗	✗
CloudPath [Mor+17]	Deployment time, latency	Functions	From store to surrogate	✗	○ (functions reuse the container)
COMET [Gor+12]	Energy consumption, latency	Code (thread)	Bytecode source and stack transferred from client	✓	✗
flexEdge	**Energy consumption, latency**	**Microservices**	**From store to surrogate**	✓	✓

TABLE 7.1: COMPARISON OF OFFLOADING SYSTEMS

7.2.3 Serverless Computing

Serverless Computing—sometimes called *Function as a Service* (FaaS)—is a delivery model for computing services that—similar to our framework—automatically manages the provisioning, execution, and scaling of services [Bal+17]. These services are implemented as stateless functions and often referred to as *lambda functions* [Hen+16].

Serverless Computing is predominantly a way to deliver Cloud Computing services. This is reflected by various commercial offerings, such as Amazon's *AWS Lambda*[2] or Google's *Cloud Functions*[3]. Recently, Cicconetti et al. [CCP19] proposed a serverless platform that extends to the edge. However, they retain certain inefficient communication mechanisms, such as every request being passed through a dispatcher without direct client-to-surrogate communication. As of today, because of its scalability, Serverless Computing is mostly used for tasks that are massively parallel [Jon+17].

While Serverless Computing overlaps with our approach in the sense that code is made available on the computing infrastructure prior to being requested, there are some fundamental conceptual differences. For example, in Serverless Computing there is no sharing of service instances across different users. Another difference is that in current commercial offerings, users of Serverless Computing platforms have virtually no control over the underlying mechanisms from an operational perspective. For example, the lifetime of the service instance is fixed by the serverless provider [Hel+19] and placements are agnostic towards dependencies on other services or data [Wan+18c]. Some serverless runtimes like *Snafu* [Spi17] do not consider the chaining of functions and heterogeneity of the hardware on which they are executed.

In contrast, our Edge Computing framework allows for a flexible definition and adaptation (via monitoring of requests) of the instance's lifetime (see Section 7.4.2.b). Furthermore, the placement decisions for data (see Chapter 9) and computations (see Chapter 8) we propose later in this thesis take into account the heterogeneity of Edge Computing hardware and networks.

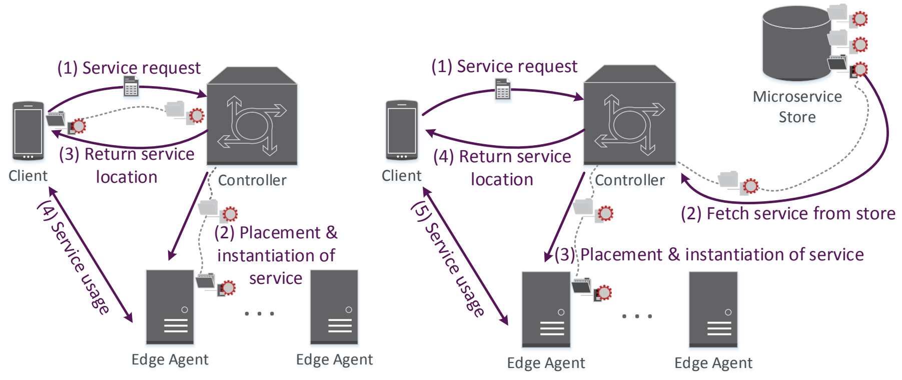
7.3 Microservice-Based Edge Onloading

We envision edge-enabled applications to rely on a repository of microservices (the *microservice store*) in order to avoid the (prior) transfer of code and execution environments. Instead of prior transfers over potentially low-bandwidth connections, microservices are fetched from the microservice store, which is assumed to be well-connected to the controller. Figure 7.1 contrasts these two approaches of traditional computation offloading (Figure 7.1(a)) versus our proposed approach (Figure 7.1(b)).

Traditional offloading includes the transfer of the application logic and execution environment in conjunction with the request of the client (step 1 in Figure 7.1(a)). Note that instead of issuing the request to the controller, it could also be directed to an agent directly. This, however, does not impact the benefit of our proposed approach. For a better comparison, the figure contrasts the cases where both approaches include a controller. Contrary to current offloading approaches,

[2] https://aws.amazon.com/lambda/ (accessed: 2020-03-05)

[3] https://cloud.google.com/functions (accessed: 2020-03-05)


(1) Service request
(3) Return service location
Client
Controller
(4) Service usage
(2) Placement & instantiation of service
Edge Agent
Edge Agent
(1) Service request
(4) Return service location
Client
Controller
Microservice Store
(2) Fetch service from store
(5) Service usage
(3) Placement & instantiation of service
Edge Agent
Edge Agent

(a) Traditional offloading (b) Store-based onloading

FIGURE 7.1: COMPARISON OF APPROACHES

we propose the novel concept of *store-based microservice onloading*, with support for chained functions and for the sharing of service instances between multiple users. Similar to previous works [Esp+17; Bha+16], we define *onloading* as pulling a requested service from a backend and instantiating it on a target surrogate. In our approach (depicted in Figure 7.1(b)), a client first issues a request for a service (step 1). Because we want to decouple the mobile device from the agents, requests for microservices are sent to the controller. Contrary to the traditional approach, the microservice is fetched from the store (step 2) and instantiated on an edge agent (step 3). Alternatively, the controller can omit the former two steps if the requested service is already running. The controller then forwards the service location (e.g., the IP address and port number from which it can be accessed) to the client (step 4). Details about the functioning of the individual steps will be described in more detail in Section 7.4.

7.3.1 Microservice Definition and Structure

Our onloading units are microservices, i.e., independent parts of an application. These services carry out tasks that often are computationally intensive and that can be composed into more complex applications. The functionality provided by a microservice is not linked to a specific application and microservices are independent in the sense that they can be executed autonomously. From an operations point of view, the individual services are developed and maintained independently from the applications that use them.

In our system, a microservice is composed of its code and the metadata that is required for the target execution environment (e.g., for building a container) and the management of the service by our Edge Computing framework. Both parts are packaged and shipped as a CSAR[4] file, an established *TOSCA* standard for the packaging of cloud services (see Explanation 7.1). Using this standard allows for the integration of our microservices into other runtime environments, e.g., as provided by the *OpenTOSCA* initiative[5]. Figure 7.2 shows an example of the unpacked

[4] Cloud Service Archive
[5] https://www.opentosca.org/ (accessed: 2020-05-27)

```
object_detection.zip
    |-object_detection
        |-decorators
        |-object_detection
        |-services
        |-utils
        |-app.py
        |-Dockerfile
        |-requirements.txt
    |-object_detection.yaml
    |-TOSCA-Metadata
        |-TOSCA.meta
```

FIGURE 7.2: CSAR STRUCTURE FOR THE OBJECT DETECTION MICROSERVICE

file and folder structure of a microservice that performs object detection (see Section 7.5.1). Files and folders related to the microservice code are colored in brown, with the exception of the file that provides the entry point for the microservice execution (`app.py`), colored in pink. In this example, we ship the service as a Docker container and the two required files to build the container are colored in turquoise (`Dockerfile` and `requirements.txt`).

Files that contain metadata for our Edge Computing framework are shown in red. The file `TOSCA.meta` contains entry information for processing the file that serves to describe the service (`object_detection.yaml`). For the description of microservices, we extend the TOSCA standard.

> EXPLANATION 7.1: TOSCA AND CSAR
>
> The *Topology and Orchestration for Cloud Application* (TOSCA) language is an OASIS (Organization for the Advancements of Structured Information Standards) standard for the description of cloud services and applications. It allows to model entities and their dependencies in YAML, a human-readable description language. TOSCA furthermore describes a file format to package descriptions. These so-called CSAR files (Cloud Service Archive) are ZIP files that contain TOSCA description files, the service itself, and additional data.

Specifically, we use the *TOSCA Simple Profile v1.1*[6] as a basis. This standard for service description is aimed at describing cloud services and, therefore, misses several properties that are relevant in the context of our Edge Computing framework. Specifically, we add properties that are required to make informed placement decisions (e.g., by considering the resource requirements of services), manage the lifecycle of services (e.g., by specifying how long they should remain active and on which ports they run), and select the appropriate services (e.g., by defining the types of data they operate on). In summary, we add the following properties to the TOSCA standard for the description of microservices:

RESOURCE REQUIREMENTS | These are the requirements in terms of computing resources required by the microservice, e.g., memory.

[6] http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/TOSCA-Simple-Profile-YAML-v1.1.html (accessed: 2020-03-12)

Service category | Microservices belong to a specific category in a hierarchy (see Section 7.4.1). This attribute defines which category the service belongs to. Child categories are separated from parent categories by a forward slash (/).

Input and output types | With these attributes, the microservice specifies which data types are taken as input and what type of data is returned.

Lifecycle management | The attribute *alive_time* defines the default time that a microservice should stay active on the edge agent. Contrary to common serverless platforms, where the lifetime is either limited to a single function execution or fixed by the provider (e.g., AWS Lambda currently has a maximum function lifetime of 15 minutes[7]), this default lifetime can be overridden by the client in order to adapt it to the application and expected request patterns. In addition, we support the idea of *polling* to monitor a microservice's activity and prevent early shutdowns or unnecessary restarts. The microservice can send an *alive* message to its agent to notify it of its activity status. This message in turn resets the alive timer of this microservice to prevent its automatic shutdown, thus allowing for more efficient use of resources and avoidance of cold starts (see Section 7.4.2.b for details).

Execution environment | We also include properties that are specific to the execution environment in which the service will be run. In our prototype implementation, we use Docker containers to ship and execute the microservices (see Section 7.5). Additional properties allow defining the port numbers of the container and network bridges.

The full TOSCA extension for the description of microservices can be found in Appendix B. These properties are required by both the client (e.g., to choose an appropriate service according to the category and input/output types), and the Edge Computing framework (e.g., to make placement decisions based on resource requirements). An exemplary microservice description is shown in Listing 7.1.

Listing 7.1: TOSCA description of a microservice

```
tosca_definitions_version:
    tosca_simple_profile_for_microservices_1_0_0
description: Template for a object detection application.
topology_template:
  node_templates:
    object_detection:
      type: tosca.nodes.microservices.docker_container
      properties:
        id: od01
        name: object_detection
        container_port: 5000
        mem_requirement: 1000
        directory: object_detection
        inputs: [ image ]
        outputs: [ image ]
        category: /images/computer_vision/object_detection
        alive_time: 1800
```

[7] information taken from https://aws.amazon.com/lambda/faqs/ (accessed: 2020-06-01)

7.3.2 Service Chaining

Applications often offload more than one of their components. In many cases, these components are executed subsequently, i.e., the output of one service is the input for the next service [Car+16; CDO19; BI14]. Therefore, services often form a *service chain* or *service pipeline*—we use these two terms interchangeably. This workflow is common to many types of Edge Computing applications, such as the processing of sensor data (e.g., first the raw values are filtered according to a threshold, then the values are aggregated), or in video analytics (e.g., first the frame is compressed, then objects in the frame are detected and annotated before aggregating the count of each object). We reflect this common workflow for Edge Computing applications by providing application developers the means to not only request the execution of an individual service, but of entire service chains.

Requesting the execution of a service chain is done by submitting a *chain description* file to the controller, which parses it and makes the services available on the edge agents (see Section 7.4.2 for details). Similar to the description file of a single microservice, the chain description is implemented as an extension to the TOSCA standard. The complete extension can be found in Appendix C. In the chain description file, the individual microservices and their properties are listed. Microservices are referenced by their store identifier. With this identifier, the properties of the individual service as defined earlier can be retrieved, e.g., for the controller to make placement decisions (see Section 7.4.2.a). In addition, new attributes are introduced for the service chain. Normally, instances of a microservice can be shared among multiple clients. If a client wants to use a service instance exclusively, i.e., enforce the creation of a new instance, the attribute *new_instance* can be set for that microservice. A *requirements* array indicates which input is required for other microservices. This also allows for branching of services, i.e., processing flows where one service requires the output of more than one service, or where the output of one service is used as input for more than one service. Furthermore, the attributes *first_in_chain* and *last_in_chain* indicate if a service is first or last in the chain. Figure 7.3 shows an example of such a branched microservice chain. Next to each microservice, an excerpt of the TOSCA description containing the relevant parts for the chaining functionality is shown. In Section 7.4.4, we will detail how we realize chained services using distributed message queues.

7.4 Functional Concept

While the previous Section 7.3 described our general approach to microservice-based onloading of services and service chains, this section describes the *functional concepts and components* that are required to realize this approach. We present the design of *flexEdge*, a distributed Edge Computing framework. The main components of flexEdge were shown in Figure 7.1(b). In the following subsections, we describe them in detail.

From the developer's point of view, flexEdge largely abstracts away most operational concerns such as placement, lifecycle management, and communication of microservices. From the end user's point of view, flexEdge can leverage proximate computing resources to offer low-latency computing services.

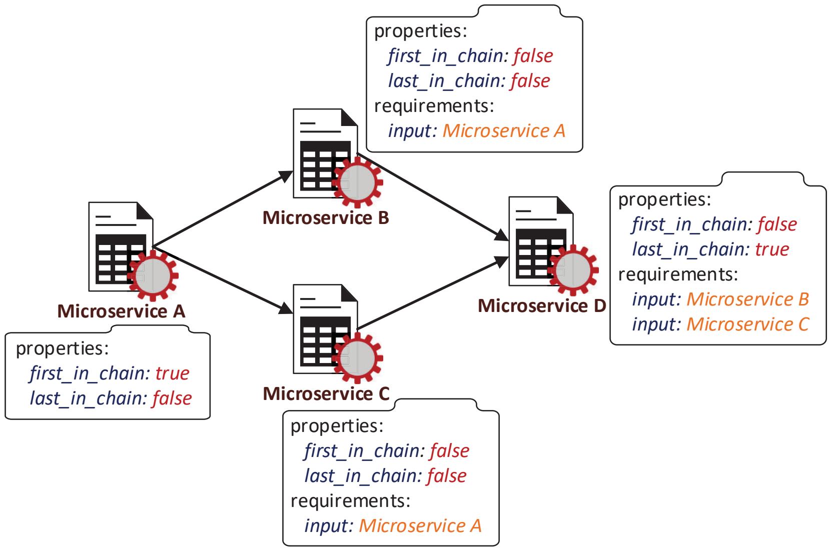


FIGURE 7.3: ILLUSTRATION OF BRANCHING IN A MICROSERVICE CHAIN

7.4.1 Microservice Store

The microservice store serves as the repository where services are uploaded to and made available by developers. The microservice store is modeled as a database and the microservices and their attributes are the records contained in the database. When a controller needs to instantiate a microservice on an agent, it requests the corresponding service from the store.

An entry for a microservice consists of a unique ID, the service name, a description, the category of the service, the types of its inputs and outputs, and the CSAR file into which the service is packaged. While the ID in the store is unique to one microservice (defined, for instance, manually by assigning a set of IDs to a developer, or automatically using UUIDs), the services themselves can be distributed in two ways. First, in larger-scale implementations, the microservice store would likely be partitioned across multiple databases. Second, since the microservice store only provides a blueprint for the controller, multiple instances of one service can be active on different agents at runtime. The unique ID described here is a global identifier that developers use to request the execution of a particular service.

To model the different categories of microservices, we chose a hierarchical, inheritance-based model because it allows for a compact and natural representation. Furthermore, this approach corresponds to the object-oriented paradigm that is common knowledge among developers. A category may be the child of a parent category and the parent of several subcategories. For example, the category of our object detection microservice (see Section 7.5.1) can be seen in Listing 7.1. This microservice belongs to the category *object_detection*, which in turn is a subcategory of the categories *computer_vision* and *images*. The information about the category, the inputs, and the outputs are especially important since they serve as the basis for our second microservice addressing scheme that is based on the semantic description and allows for an automatic selection of services (see Section 7.4.2.a).

7.4.2 Controller

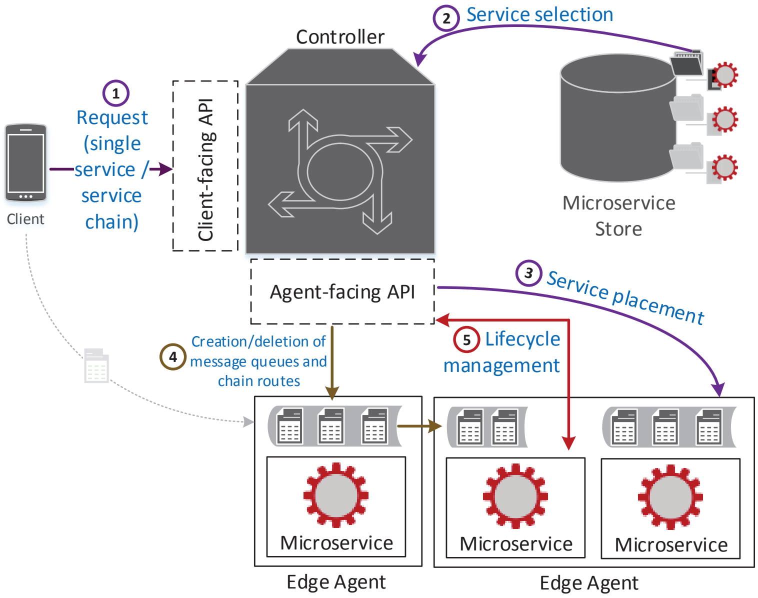


FIGURE 7.4: OVERVIEW OF THE CONTROLLER'S FUNCTIONALITIES

The controller is the centerpiece of flexEdge. Figure 7.4 depicts the main functionalities of the controller. The controller maintains a list of edge agents that execute the microservices on the surrogate machines. Furthermore, the controller has a global view on the system, including which microservices are currently running on which agents. It has two main APIs, one facing the clients, and one facing the agents.

The controller's functionalities can be divided into three parts. First, following requests from clients (labeled ① in Figure 7.4), it selects an appropriate microservice for the request (② in Figure 7.4), and places the service(s) on edge agents (③ in Figure 7.4). Section 7.4.2.a describes this part of the functionality in more detail. Second—following the placement decision—the controller is responsible for the creation and deletion of the message queues associated with the microservices instances (④ in Figure 7.4). Section 7.4.4 details the implementation of message queues in flexEdge. Third, the controller manages the lifecycle of the microservice instances (⑤ in Figure 7.4), i.e., it monitors their execution and decides when to stop services. Section 7.4.2.b describes this lifecycle management in more detail.

7.4.2.a Service request and placement

Clients request either the execution of a single service or of a service chain. Both types of requests are issued to the controller via a call to a REST-style API. The execution of a single service can be requested in two ways: (i) by providing the store identifier of the microservice or (ii) by specifying a category and the data types of the inputs and outputs. Based on this information the controller selects a matching microservice from the store. In case the client requests the execution of a service chain, it submits the chain description (see Section 7.3.2) to the controller. In both cases, the controller sends a response back to the client, detailing the location of the service(s) (i.e., the corresponding agent IP addresses and exposed ports), and the queue name where requests can be issued to.

Conceptually, our approach of selecting services relies on mechanisms that are similar to UDDI[8] [Cur+02]. UDDI is a specification for the discovery of web services that consists of three main elements: white pages (containing the name and meta information about the service), yellow pages (describing the category of the service), and green pages (containing technical information about the service and further details on how to access it). Mapped to our functional design, white pages and yellow pages are represented by the attributes of the service as saved in the database record of the microservice store (i.e., the name, description, and category of the service), while technical details (comparable to green pages) are contained in the TOSCA description file (e.g., specifying the container runtime or the ports).

After receiving requests, the controller needs to decide (i) if running instances of services should be reused and (ii) on which agents to place new microservice instances. For these placement decisions, different approaches can be employed. This chapter is intended to describe the overall design of our microservice execution mechanism, and hence, concrete placement algorithms are beyond the scope of this chapter. The controller's global view (e.g., w.r.t. the resources available on the agents and where microservices are running) and the description of the services (e.g., their resource requirements) enable the implementation of different placement strategies on the controller. Chapter 8 will present a strategy that could be implemented at a controller for the placement of microservices.

7.4.2.b Service lifecycle management

When a microservice is requested, we can distinguish between a warm start and a cold start. A cold start of a microservice happens when the microservice to be used is not yet running on the agent. In a cold start—following the placement decision of the controller—the microservice has to be transferred from the store to the agent and then started on the agent. In contrast, a warm start of a microservice happens when the client uses an already running instance of a microservice. In this case, the process of transferring the service from the store and starting the service will be skipped, i.e., step 2 and step 3 in Figure 7.1(b).

Besides the placement of microservices, the controller is also responsible for terminating services. By default, this is done after the default alive time specified in the service description has elapsed. The framework also offers the possibility to override this at the time of service instantiation. In that case, microservices report to the controller when they are being actively used, e.g., when a certain number of requests have arrived in a time frame. The exact timing is left at the discretion of the microservice developers, who need to actively implement this feature. Whenever the controller receives such a *polling message* from the service, the service lifetime is reset to the initial value. Timer tasks running on the controller regularly terminate services whose lifetimes have expired. This mechanism allows for a flexible management of the service lifetime. This dynamic termination strategy is in contrast, e.g., to the current practice in Serverless Computing, where function instances are terminated after a fixed time. Manner et al. [Man+18] have investigated influencing factors on the cold start latency in the domain of Serverless Computing. They

[8] Universal Description, Discovery, and Integration

furthermore concluded that there is a significant difference between the latency that the user *perceives* versus the actual, *billed duration*. In Edge Computing, where many offloaded application components represent user-facing functionality (e.g., in the domain of rendering or computer vision), this insight is especially important.

Clearly, lifecycle management incurs a tradeoff. Terminating services too early (i.e., when clients still frequently request them) means more cold starts, and hence, higher end-to-end latencies as will be shown in Section 7.6.2.a. On the other hand, keeping services active consumes scarce resources on the edge agents. We leave the exploration of this tradeoff for future work.

7.4.3 Edge Agent

The edge agent is a background service that runs on the surrogates and enables the actual execution of microservices. It does so by providing two interfaces: (i) a *northbound interface* to the controller, and (ii) a *southbound interface* to the execution environment. We borrow this terminology from the domain of SDN (see Section 3.5.3), where it is used to denote the distinction between interfaces to a high-level control entity (northbound), and the execution of the controller's policies (southbound). In our system, the definition of these interfaces serves to decouple the control functionalities, e.g., placement and monitoring, from the concrete execution environment on the agent.

The edge agent is designed to be extensible in order to support different execution environments. One example of such an execution environment is the container platform Docker (see Explanation 7.2 in Section 7.5 for details). The agent manages the execution environment for the microservices (e.g., by issuing commands to the Docker command line interface for the starting and stopping of containers) and interacts with the controller for their lifecycle management (e.g., it receives requests from the controller to stop a running service). The agent reacts to instantiation requests coming from the controller. If the corresponding container image (see Section 7.5 for details about the container runtime) does not exist on the agent yet, it will be built upon requesting the service. Future invocations of microservice instances will use the pre-built image unless the client sends a *force_rebuild* flag. Figure 7.5 illustrates the role of the edge agent with the example of Docker as an execution environment.

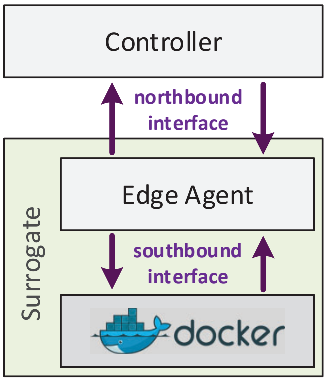


FIGURE 7.5: EDGE AGENT AND ITS INTERFACES

7.4.4 Message Queues

Because we envision applications to be broken up into individual microservices, these microservices need a way to communicate, (i) with the client to receive inputs and deliver results, and (ii) between each other to realize service chains (i.e., the result of one microservice is forwarded as input to the next microservice).

In flexEdge, we realize this communication through distributed message queues. Distributed message queues function through a *message broker* that relays messages between senders and receivers, often allowing for *publish-subscribe* communication patterns [Eug+03], which have been used extensively, e.g., in the IoT domain [Sil+16; Wan+12]. In our dynamic edge environment, using message queues is advantageous for the following reasons:

(i) Asynchronous communication | Message queues provide an asynchronous communication mechanism. Hence, clients are able to issue non-blocking requests to the queue and do not have to wait for the completion of the request, enabling them to continue to execute local parts of an application. In contrast, other offloading frameworks use synchronous communication patterns. As an example, *MAUI* [Cue+10] uses RPCs[9] between the clients and the surrogates.

(ii) Decoupling | Having a message broker in-between the clients and the microservice instances decouples those two entities. This naturally supports the design of our system, in which microservice instances can be shared among different clients. The decoupling furthermore facilitates future optimizations of our system. For instance, supporting the scaling out of instances can be done by dynamically re-assigning requests to queues that correspond to different service instances. In comparison, many other offloading frameworks, e.g., [LWB16; Chu+11; Kos+12] cannot leverage these advantages, as they expect a direct relationship between the client and the offloaded functionality.

(iii) Additional guarantees | Depending on the message broker that is used, it can offer additional guarantees, such as delivery guarantees. Those are relevant, for instance, when an edge node fails and requests need to be retransmitted to other service instances.

In our design of the message queue concept, each microservice is associated with a *request queue*, addressed by a unique name. Furthermore, each service chain has a distinct *result queue*, to which the result of the last service in the chain is pushed. The message queues are distributed in the sense that each agent runs a message broker that maintains the message queues of each service running on that agent. After the controller has made placement decisions for the individual services of the chain, it constructs a *route*, containing the chain's structure. This route is a python dictionary that is passed through the entire service chain in the message header, such that microservices in the pipeline know to which queue they should publish their results. In detail, the chain route contains the following information: (i) the chain's first message queue where the client pushes requests (denoted by the key `start`), (ii) the IP addresses of the agents where a given service is placed (since each agent has its own message broker), (iii) ports (container-internal and exposed to the host) that the service listens to, (iv) the previous and next message queues (denoted by the keys `prev` and `next`) (v) the identifier of the result queue (denoted by the key `result`), and (vi) a unique identifier of the chain (`chain_id`).

[9] remote procedure calls

Listing 7.2 shows an example of such a route (serialized to JSON), consisting of three service instances in the following order: `image_compression_1` → `super_resolution_1` → `mesh_construction_1`.

LISTING 7.2: EXAMPLE OF A MICROSERVICE CHAIN ROUTE

```
{
'start': 'image_compression_1',
'result': 'result_27fceda6-d5ea-4461-bbd2-5cb8ae4cdb0f',
'image_compression_1':
        {
        'agent': '18.189.1.228',
        'ports': [{'container_port': 5000, 'host_port': 59203, '
            ↪ protocol': 'tcp'}],
        'next': ['super_resolution_1'],
        'prev': []
        },
  'super_resolution_1':
        {
        'agent': '18.189.1.228',
        'ports': [{'container_port': 5000, 'host_port': 61712, '
            ↪ protocol': 'tcp'}],
    'next': ['mesh_construction_1'],
        'prev': ['image_compression_1']
        },
'mesh_construction_1':
        {
        'agent': '18.189.1.228',
        'ports': [{'container_port': 5000, 'host_port': 56974, '
            ↪ protocol': 'tcp'}],
  'next': ['result_27fceda6-d5ea-4461-bbd2-5cb8ae4cdb0f'],
        'prev': ['super_resolution_1']
        },
'result_27fceda6-d5ea-4461-bbd2-5cb8ae4cdb0f':
        {
        'agent': '18.189.1.228'
        },
'chain_id': '25cef7cf-5a73-41a4-a677-760dd3c7ef2a'
}
```

7.5 Implementation Details

We realize a prototype implementation of our proposed concept. This section describes further technical details about the implementation. Figure 7.6 shows an overview of the implemented system and its components. Our implementation consists of (i) a centralized *controller*, to which clients submit requests for the execution of microservices, (ii) *edge agents* that run the containerized microservices, and (iii) the *microservice store*. The controller and the agents are implemented as Python applications. For the containerization of microservices, we use *Docker* (see Explanation 7.2).

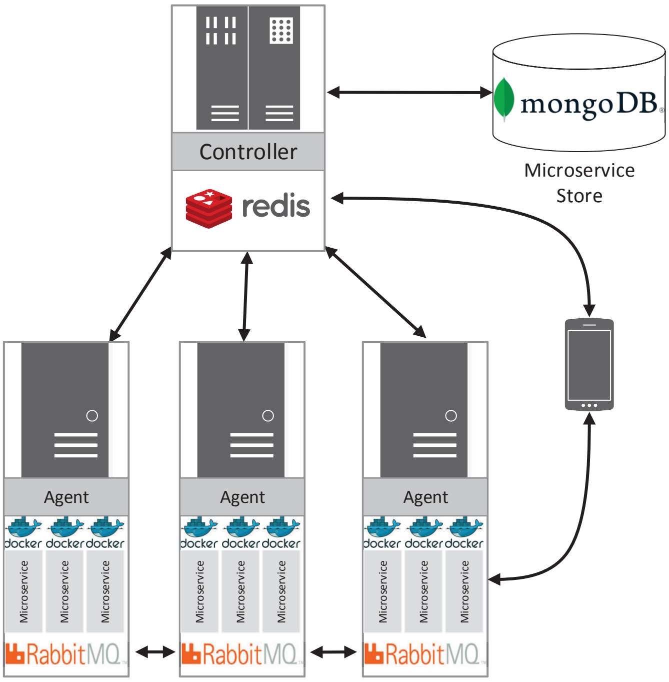


FIGURE 7.6: PROTOTYPE IMPLEMENTATION

EXPLANATION 7.2: DOCKER

Docker[a] has emerged as the predominant platform for container-based virtualization. All running containers on a system share the same kernel; hence, it is often referred to as *OS-level virtualization*. Compared to virtual machines, containers typically are smaller and hence, quicker to instantiate (see also Section 3.5.2 for details about different virtualization technologies). Docker is a platform that consists of several pieces of software. At its core is the *daemon* that builds and runs containers. Users interact with the daemon through a command-line interface. Containers in Docker are built using read-only *images* as templates. Users can pull such images from a registry and create customized images based on a base image. This is done with the help of the *Dockerfile*. This file has a special syntax to define the necessary steps for the creation of containers from a base image (e.g., by installing custom software packages). Docker offers different possibilities to interconnect containers, e.g., through bridge networks. Furthermore, internal network ports of a container can be mapped to a port of the host machine, making the container available from outside.

[a] https://www.docker.com/ (accessed: 2020-02-23)

Clients are assumed to know the controller's API endpoint to which they submit microservice execution requests. We further assume that our controller has a global and correct view of the overall system, including all available agents. In order to obtain this global view, the controller can retrieve information from a *Redis*[10]

[10] https://redis.io/ (accessed: 2020-03-04)

instance. This in-memory key-value store is kept updated with information about agents and the current state of the system (e.g., which microservice instances and resources are currently available). To parse the microservice descriptions, we adapt the TOSCA parser from the OpenStack project[11], so that it is able to parse our extensions for the definitions of microservices and service chains (see Section 7.3.1 and Section 7.3.2). Following a client request and a placement (see Section 7.4.2.a), the requested services are either newly instantiated or existing ones are reused and requests redirected to them. In the first case, the controller creates a new, unique instance name for that service, composed of its name (as found in the TOSCA description) concatenated with a sequential numbering. This unique instance name is required by clients, agents, and the controller to address the service instance. The controller also creates network port mappings between the internal ports of the microservice container and the host system. To do so, the controller assigns a host port from the range of 50001 to 61999 (since those fall into the range of so-called *dynamic ports* as defined in RFC6335[12] and hence, do not conflict with any well-known system ports). When assigning a host port, the controller also takes into account the already running microservices on the agent, as to avoid port conflicts.

All REST-style APIs are implemented using Flask[13], a lightweight web framework for Python. The edge agents also run as dockerized Python applications on the surrogates. The microservice store is connected to the controller and realized in *MongoDB*[14], a document-oriented database. Each microservice is stored as a document in a MongoDB collection. Each document has a unique ID (the *store ID* of the microservice that can be used to reference it), its name, a textual description, input and output types, and the category. Because microservices can have a substantial size, we enable *GridFS* in the database system to store the CSAR files of the microservices[15]. The document of a microservice contains a reference to the GridFS document where the corresponding CSAR file is stored. Categories for microservices are defined in a separate collection. Each document stored there contains a reference to a parent category (unless it belongs to a root category), allowing us to model hierarchies of service categories.

To realize the message queues, we use *RabbitMQ*[16], an open-source message broker. Each agent runs an instance of the message broker. It is responsible for creating and deleting message queues for each microservice running on that agent. For each microservice, a message queue is created to which requests for that service are sent. This request queue is addressed by the same identifier as the corresponding microservice instance. The result queue for a service chain is maintained on the agent that executes the first service in the chain. The identifier for the result queue is generated with a UUID[17].

7.5.1 Demo Microservices

To evaluate our system, we developed three microservices. We have made them available for the research community[18]. All microservices were implemented in

[11] https://github.com/openstack/tosca-parser (accessed: 2019-04-08)

[12] https://tools.ietf.org/html/rfc6335 (accessed: 2020-05-13)

[13] https://palletsprojects.com/p/flask/ (accessed: 2020-03-04)

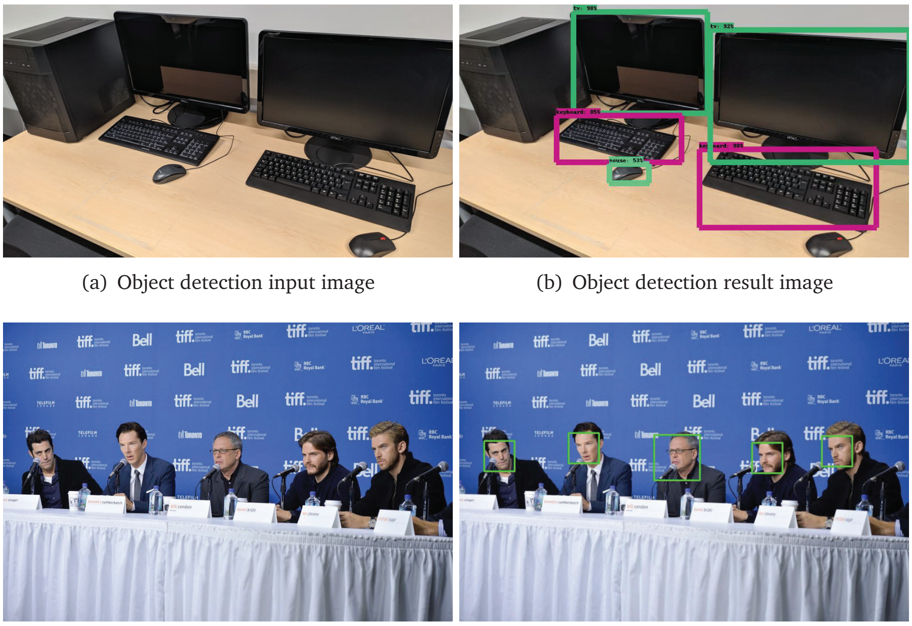
[14] https://www.mongodb.com/ (accessed: 2020-02-24)

[15] MongoDB currently limits the size of regular documents to 16MB

[16] https://www.rabbitmq.com/ (accessed: 2020-03-10)

[17] Universally unique identifier

[18] https://github.com/Telecooperation/flexEdge-microservices (accessed: 2019-11-18)



(a) Object detection input image

(b) Object detection result image

(c) Face detection input image

(d) Face detection result image

FIGURE 7.7: EXAMPLE RESULTS PRODUCED BY THE MICROSERVICES

Python, use Flask and Bottle[19] to implement a REST-based API, and were shipped as Docker containers.

(i) OBJECT DETECTION | Using TensorFlow, we performed object detection on an image. The microservice returns the original image including the detected objects, enclosed by rectangles and labeled with the name of the object and a confidence value. The microservice code was adapted from the TensorFlow Object Detection API[20] and we used the *ssd_mobilenet_v1_coco* model trained on the COCO dataset. Figures 7.7(a) and 7.7(b) show example input and output images for this microservice.

(ii) FACE DETECTION | Using OpenCV and a *LBP Cascade* classifier, our second microservice detects faces in an image and returns the original image with the faces enclosed by rectangles. An example of the output produced by this service (with Figure 7.7(c) as input) can be seen in Figure 7.7(d)[21]

(iii) WORD COUNT | Lastly, we used a simple word count application that counts the number of words in a given text file. This application served as a simple benchmarking tool for our experiments.

[19] https://bottlepy.org/ (accessed: 2020-03-30)

[20] https://github.com/tensorflow/models/tree/master/research/object_detection (accessed: 2019-11-18)

[21] Pictures are taken from the *WIDER FACE* dataset: http://shuoyang1213.me/WIDERFACE/ (accessed: 2020-02-20)

7.5.1.a Service chain

In addition, for the performance evaluation of our chaining mechanism (see Section 7.6.3), we considered a simple chain for benchmarking. The service chain takes text as input and at the first stage, a service just echoes back the input into the next queue. The second service splits up the text into individual words. The third service counts the number of individual words. Finally, the last service echoes back the result of the word count. Figure 7.8 illustrates this demo chain of microservices. The TOSCA description file that describes the chain can be found in Appendix D.

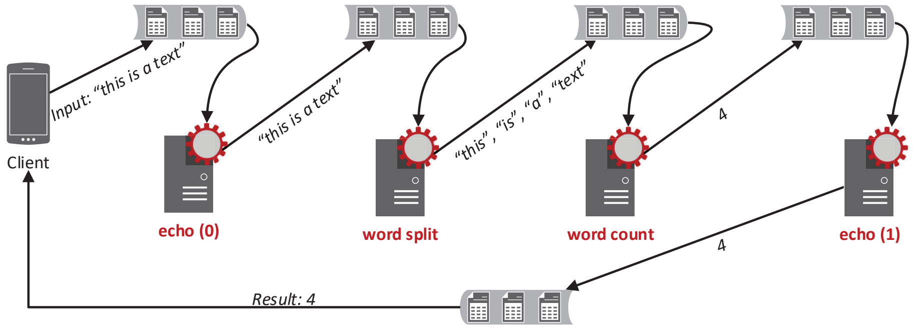


FIGURE 7.8: DEMO MICROSERVICE CHAIN FOR THE EVALUATION

7.6 Evaluation

We evaluate the store-based onloading approach of our proposed Edge Computing framework with regards to end-to-end latency (Section 7.6.2.a) and energy consumption (Section 7.6.2.b). We also provide some further discussion on the location of the microservice store (Section 7.6.2.c) and the performance and energy impact when executing the services on a mobile device (Section 7.6.2.d). Finally, we show the efficiency of our approach to chaining microservices and compare it with a state-of-the-art serverless platform (Section 7.6.3).

7.6.1 Experimental Setup

As an edge node, we use a Lenovo ThinkCentre M920X Tiny with an Intel Core i7-8700 and 16 GB RAM running Ubuntu 18.04. This device is a consumer-grade desktop computer that has a small form factor and hence, is an ideal example of an Edge Computing surrogate that might be deployed in practice. The edge node is connected via Gigabit Ethernet to a Linksys WRT 1900 AC wireless access point that at the same time serves as a 802.11nac gateway for the mobile device. Besides WiFi connectivity, we also conduct our experiments using a 4G cellular network in order to cover the different types of wireless connections that client devices encounter. We use the network of the carrier *Deutsche Telekom* with an advertised maximum bandwidth of 300/50 Mbit/s (down/up). The controller and the microservice store are colocated on the same machine, a Citrix Xen VM. The VM uses 1 Core of an AMD Opteron 6380 (clocked at a maximum of 3.4 GHz), has 8 GB of RAM and runs Ubuntu 16.04. This VM runs in the same backend network as the edge node.

As a mobile client device, we use a Google Pixel 2XL phone (8-core Qualcomm Snapdragon 835, 4 GB RAM) running Android 9.

For the OpenWhisk environment used in Section 7.6.3, we use an Amazon AWS EKS cluster, consisting of three EC2 instances of type *m5.large* (2 vCPUs clocked at up to 3.1 GHz each, 8 GB RAM, Ubuntu 18.04), located in the Frankfurt availability region. Two of those instances are used for the OpenWhisk *invoker nodes* and one for the *core node*. For a better comparison, the agents of flexEdge also run on m5.large instances for this part of the evaluation.

7.6.2 Store-Based Microservice Onloading

We use the three microservices described in Section 7.5.1 for the evaluation and use the mode where microservices are selected based on their store ID by the client. Experiments are conducted in both the WiFi and 4G cellular networks. Furthermore, we consider both cold starts and warm starts. Recall that in a cold start, there is no running instance of a microservice available and, hence, the service has to be instantiated on the agent. We compare our approach of store-based onloading with traditional offloading, in which the entire service (i.e., in our system the CSAR file) to be executed is transferred from the device to the controller/agent with every invocation (in both cold start and warm start).

7.6.2.a Latency

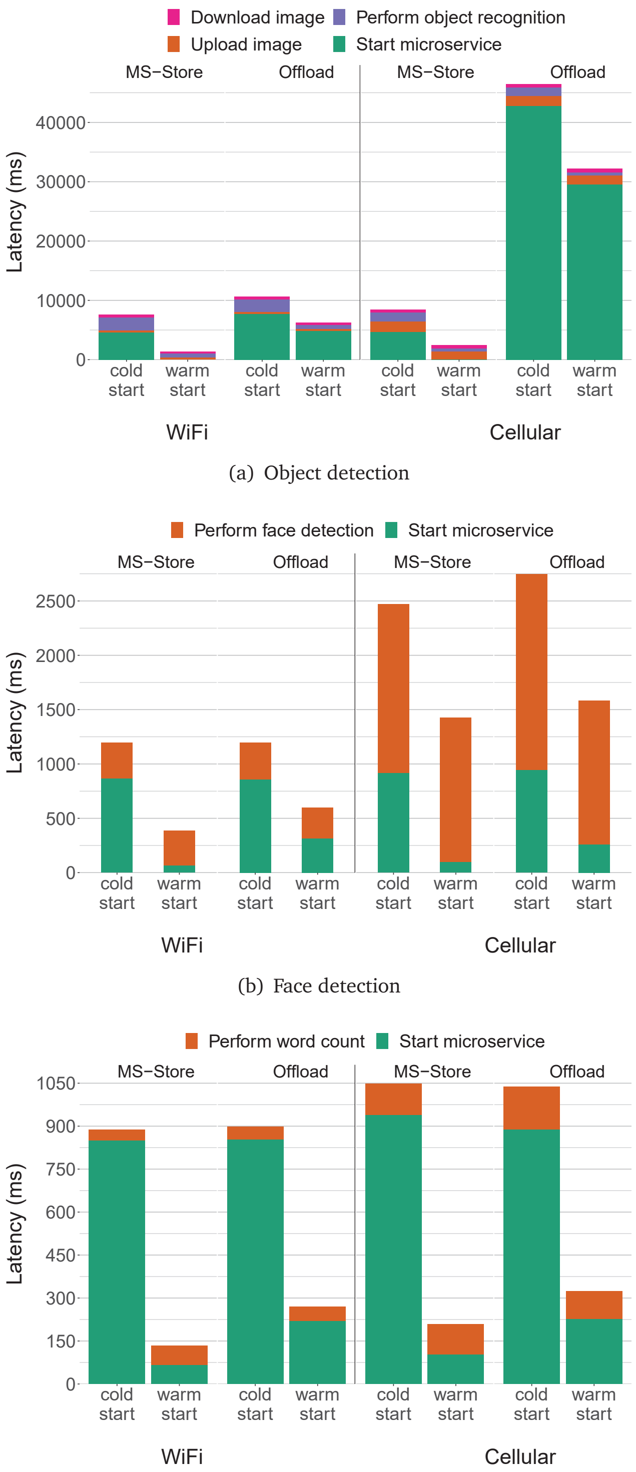
TABLE 7.2: OVERVIEW OF SPEEDUP[a]

	cold start	warm start
WiFi	OD: 1.406× (28.89 %)	OD: 4.531× (77.93 %)
	FD: 0.996× (-0.40 %)	FD: 1.549× (35.44 %)
	WC: 1.012× (1.20 %)	WC: 2.014× (50.34 %)
Cellular	OD: 5.508× (81.84 %)	OD: 13.031× (92.33 %)
	FD: 1.111× (10.02 %)	FD: 1.109× (9.83 %)
	WC: 0.990× (-1.04 %)	WC: 1.561× (35.92 %)

[a] **OD:** object detection, **FD:** face detection, **WC:** word count

First, we evaluate the end-to-end latency, i.e., the time between when the service request is sent from the client device and when the result is received. Each experiment is repeated 30 times. The mean values of the execution time are shown in Figure 7.9, plotted individually for each microservice. The detailed values for each microservice, along with the standard deviation, minimum and maximum values can be found in Appendix E.

For a more fine-grained analysis, we divide the analysis of the overall latency into two steps: (i) the time it takes to invoke the microservice and (ii) the actual execution time of the task (including the transfer of input and result data). This allows to better quantify the overheads of cold starts, i.e., the transfers of services prior to their execution and the time to instantiate them on the agents. To gain even more detailed insights on the total execution time, we choose one microservice—the object detection—for which we also measure the individual times for uploading the input and downloading the result.



(a) Object detection

(b) Face detection

(c) Word count

FIGURE 7.9: END-TO-END LATENCY[†]

Table 7.2 summarizes the average reduction times for the microservices, grouped by startup mode and type of wireless access. The table shows the absolute reduction (as a factor) and the corresponding percentage of the reduction. In conclusion, we saw an average reduction in the end-to-end latency of 1.1–14 ×. From the results, we can make a number of observations. If we use our approach in warm start, we see a reduced latency across all microservices. For a cold start, the benefit of our approach depends on the size of the microservice. To put this into perspective, the sizes of the CSAR files are 28 MB (object detection), 12 KB (face detection), and 2 KB (word count). In the conventional offloading approach that we use as a baseline, the CSAR file always has to be transferred from the client device. This explains the reduction in latency when using store-based onloading for the first step (service invocation) in the object detection and (to a lesser degree because of its size) the face detection microservice.

In two cases (FD/WiFi/cold start, and WC/cellular/cold start) our approach led to a small increase in latency. The reason for this is the very small size of those microservices, and that the delay of transferring them from the store to the agent is comparable to transferring a very small CSAR file via a cellular network. In warm start, however, even with those small services, our approach reduces the latency by 9.83 %–92.33 %. The highest reduction in warm start latency can be seen with the object detection microservice, with a reduction of 77.93 % (over WiFi) and 92.33 % (over a cellular connection).

In conclusion, the benefits of our store-based onloading were especially striking if we assumed that large microservices would have to be transferred from the mobile device to the target execution environment. Furthermore, we assumed that even in warm start, this would have to be performed in traditional offloading. Our largest microservice is a representative example of a functionality that would be carried out outside the mobile device, and therefore, this demonstrates the practical benefit of our approach for executing services outside a client device. For smaller services, however, the benefit of onloading is substantially smaller.

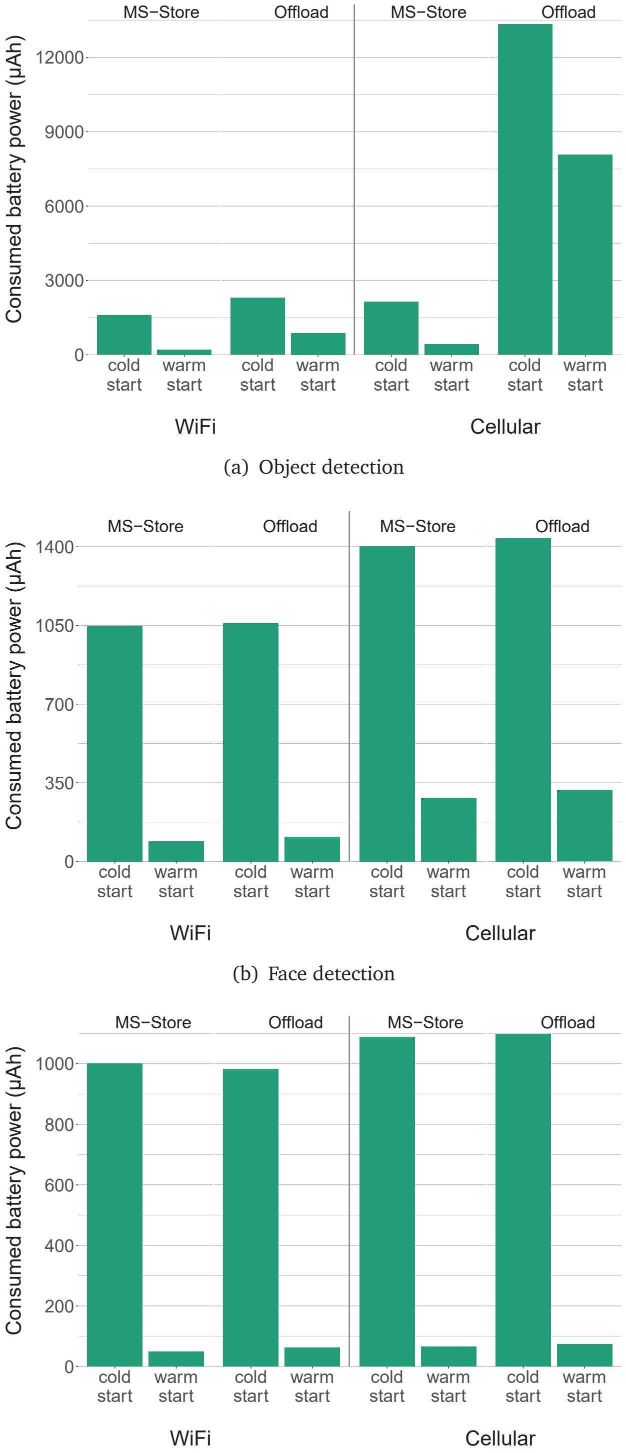
7.6.2.b Energy consumption

TABLE 7.3: OVERVIEW OF ENERGY SAVINGS[a]

	cold start	warm start
WiFi	OD: 1.444× (30.73 %) FD: 1.012× (1.22 %) WC: 0.982× (-1.81 %)	OD: 4.305× (76.77 %) FD: 1.231× (18.79 %) WC: 1.265× (20.96 %)
Cellular	OD: 6.247× (83.99 %) FD: 1.025× (2.48 %) WC: 1.008× (0.84 %)	OD: 18.850× (94.69 %) FD: 1.123× (10.99 %) WC: 1.140× (12.25 %)

[a] **OD:** object detection, **FD:** face detection, **WC:** word count

We now show how our approach benefits the mobile device in terms of prolonging its battery life. To quantify this benefit, we measure the energy consumption of our store-based onloading and compare it with traditional offloading. To measure the energy consumption of the smartphone, we use the value of the `BATTERY_PROPERTY_CHARGE_COUNTER` property reported by the *Android Battery*



(a) Object detection

(b) Face detection

(c) Word count

FIGURE 7.10: ENERGY CONSUMPTION[†]

Manager. This property can be accessed via the Android developer API and provides the remaining battery capacity in μAh. By preliminary testing, we found that the phone's operating system updates the battery manager every 1.5 seconds. A single run in our experiments (i.e., executing a microservice once) can however take less than a second. Therefore, we consider the difference in battery capacity across all runs and divide this delta with the number of experiments to get an estimation of the consumed power for a single execution.

Figure 7.10 shows the results of the consumed battery power for a single execution, averaged from all 30 measurements. Comparing the results with those of the latency evaluation in the preceding Section 7.6.2.a, we can observe a correlation between the reductions in the end-to-end latency and the power consumption. Similarly, how big the benefit of our approach is depends both on the size of the microservice and whether it is invoked in cold or warm start. When invoking a service in warm start, we see a reduction in the energy consumption for all microservices. How big this reduction is depends on the size of the microservice. In the traditional offloading approach, transferring a larger microservice CSAR file consumes more energy, as can be seen in the example of the object detection. For a cold start invocation, the energy savings also depend on the size of the microservice, but overall, the benefits are less striking (in this case, the mobile device has to wait for the instantiation and keep the connection open and hence, energy is consumed in the meantime). For one case—the word count microservice invoked in cold start over a WiFi connection—we found a negligible increase in energy consumption when using our approach. We attribute this to measurement inaccuracies in the internal battery manager software.

As in the previous section, we summarize the average savings of our approach in Table 7.3. If we compare this table with Table 7.2, we can see that the results are similar, i.e., our approach is most beneficial with the object detection microservice, which is the largest in size. In conclusion, this part of the evaluation showed that overall our approach is able to save battery life for the mobile device.

7.6.2.c Microservice store location

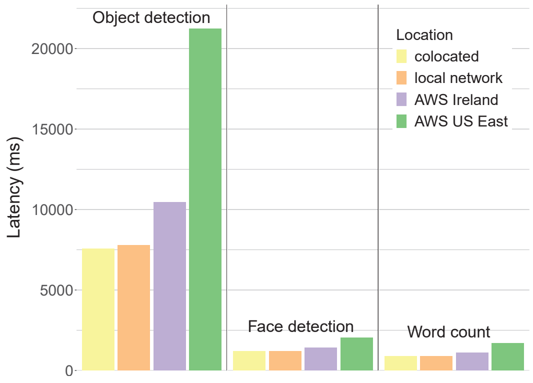


Figure 7.11: Impact of the store location on the latency†

In the previous experiments, the microservice store was assumed to be colocated at the controller and close to the agents (more specifically, in the same local area network). However, if we envision a large-scale deployment of our system, we can make two observations. First, we would not have a single controller but multiple distributed controllers. Those could for example form a hierarchy, in which each controller is responsible for one (geographic) region. Second, a single controller is unlikely to be hosted on hardware that has the capacity to host all microservices. Hence, in practice, we would not only see a decoupling of the controller and the microservice store, but multiple instances of those two entities. Having the microservice store not colocated at the controller potentially incurs a big performance penalty, e.g., when the store is located in a distant cloud infrastructure.

We now quantify the impact of the microservice store location on the cold start latency and derive recommendations for a large-scale deployment of flexEdge. We consider the microservice store to be (i) colocated on the controller, (ii) in the same local network, and (iii) in two cloud locations. For the latter, we use Amazon AWS EC2 instances located in the availability regions US East and Ireland. In the experiment, the controller and the agent are both located in Darmstadt, Germany.

The results are shown in Figure 7.11. Depending on the size of the microservice, the store location greatly affects the latency. While a store located in the same network has a relatively small impact on the latency, using Cloud services increases the latency by 33 % (AWS Ireland) up to 200 % (AWS US East). Hence, for a viable deployment, the microservice store should be close to the controller and agents so as not to jeopardize the benefits of our offloading scheme. However, in practice, the microservice store, controller and agents would likely be well-connected via wired networks, compared to more unreliable (and sometimes metered) wireless networks that mobile clients use. Hence, even with additional network hops to transfer the microservices to the agents, our approach is beneficial for the overall latency.

7.6.2.d Comparison with local execution

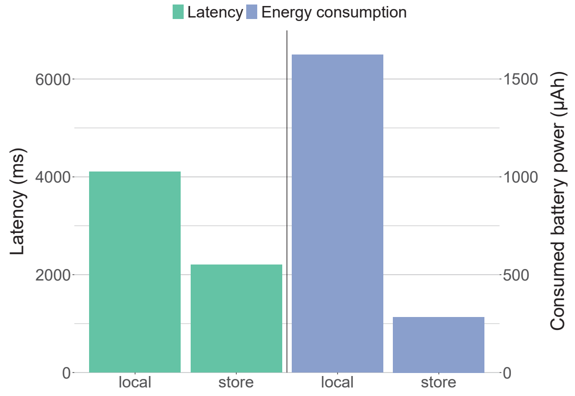


FIGURE 7.12: COMPARISON WITH LOCAL EXECUTION†

We now compare the difference in end-to-end execution latency and energy consumption if a service is executed locally on the mobile phone. For this, we use the object detection as an example and implement a version of this microservice for

Android. The code is adapted from the TensorFlow Android Camera Demo[22]. We use *faster_rcnn_inception_v2_coco* as a model. In this experiment, we use a warm start execution through WiFi with the same hardware as described before.

Figure 7.12 shows the result. Our approach leads to a reduction in latency of about 50 % compared to local execution. Furthermore, the consumed energy of the local execution is almost six times higher than the consumed energy of the offloaded execution through the microservice store. These results demonstrate that our store-based microservice onloading can help in prolonging the mobile device's battery life while reducing the end-to-end latency (and hence increase the user's quality of experience).

There is, however, a tradeoff between those gains and the overhead of onloading. As we have seen in Section 7.6.2.a and Section 7.6.2.b, for very simple microservices (e.g., the word count example) the overhead introduced by using flexEdge (i.e., issuing a request to the controller, fetching and instantiating the service or redirecting the user to the running service) can outweigh the benefits. In the previous sections, we have shown how these benefits differ with varying microservice size. For very simple microservices, executing them locally is a viable alternative to either offloading or store-based onloading. Hence, in future use of flexEdge, a careful partitioning of applications is required to take full advantage of our approach. Our contribution in this chapter is meant to provide a basis for an alternative approach to traditional offloading, which can be used following existing partitioning strategies.

7.6.3 Performance of Chained Services

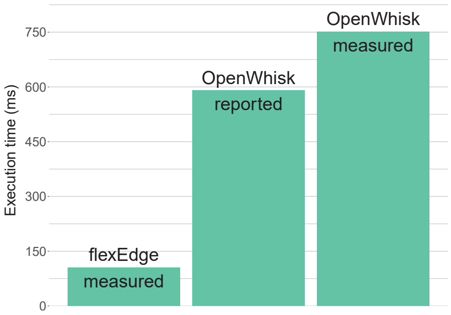


FIGURE 7.13: PERFORMANCE COMPARISON OF CHAINED SERVICES†

Lastly, we evaluate the performance of the chaining feature in flexEdge, where multiple microservices are executed subsequently. We measure the total time it takes to execute the entire service pipeline and return the result. We compare our approach in which microservices are chained through a distributed message queue (see Section 7.3.2) with the chaining as realized in *OpenWhisk*[23], a state-of-the-art serverless platform (see Explanation 7.3 for details).

[22] https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android (accessed: 2019-02-19)
[23] https://openwhisk.apache.org/ (accessed: 2020-02-17)

> EXPLANATION 7.3: OPENWHISK
>
> Apache OpenWhisk is a serverless cloud platform that executes functions in response to requests or events (*triggers*). Its general architecture resembles flexEdge, e.g., *invoker nodes* in OpenWhisk are similar to flexEdge agents, and the OpenWhisk *core node* performs similar functions as our controller. OpenWhisk executes *actions*, which are stateless functions that can be written in different languages, in containers that can be managed with different frameworks (e.g., *Kubernetes* or *Mesos*). Each invocation of an action produces an *invocation record* that contains the function's results as well as metadata about the invocation, e.g., the execution time. Both function inputs and outputs are mapped to key-value pairs. The subsequent invocation of different functions is called *action sequence* in OpenWhisk.

Comparison of approaches. OpenWhisk stores every intermediate result of an action (OpenWhisk terminology for a function) in an auxiliary database. For the next function in the chain, the controller has to first fetch this result before passing it as input to the subsequent function. Furthermore, OpenWhisk limits the size of the data that can be passed in this way to 1 MB. For a lot of applications, this would clearly not be sufficient. In case the size of the result exceeds the maximum value, results have to be stored in auxiliary storage and a reference to the storage location can be returned by the action. Selecting suitable means for ephemeral storage in serverless platforms is an ongoing field of research [Kli+18]. Even disregarding this restriction, we expect our approach to be much faster because intermediate results are not passed through the controller or temporarily stored. Instead, each service is given its chain successor as input and directly pushes the intermediate result into the input message queue of that successor.

Performance comparison. As a benchmark to compare the chaining of services, we use the word count chain as described in Section 7.5.1.a. For the input, we use a pre-generated filler text with a size of 683 KB. For both flexEdge and OpenWhisk, we use a warm start invocation of services. Figure 7.13 compares the averaged total execution times of 10 runs. For OpenWhisk, we perform two measurements. First, we start the chain via a blocking command line request and record the execution time with the Unix *time* command (labeled *OpenWhisk measured* in the plot). Second, OpenWhisk itself collects statistics about the runtime of actions (labeled *OpenWhisk reported* in the plot). The slight differences in measurements occur because OpenWhisk only starts measuring once the first action is invoked. Our measurements include the overhead of OpenWhisk's own message queue, to which requests are sent before the action is invoked.

Result interpretation. The results clearly show the advantage of our chaining approach, with an execution time of the pipeline that is almost six times faster compared to the reported execution times by OpenWhisk. For our own measurements of the OpenWhisk execution time, the difference is almost 7 ×. Since the payload of messages in our experiment does not exceed 1 MB (which would cause OpenWhisk to move the results to auxiliary storage), the reduction in execution time is solely due to the fact, that our approach does not move intermediate results to the controller before passing them to the next microservice. Note that in our experiment here, we considered only two agents/invokers that along with the controller were colocated in the same local backend network. In more large-scale deployments this might not be the case, rendering the advantage of our approach even more important. The actual impact of function chaining is intertwined with the placement decision of the services in the chain. For example, in our experiment reported here, we noticed that while our approach placed all microservices on the same agent (according to the colocation strategy presented in Section 7.4.2.a), OpenWhisk placed the *echo* and *word count* actions on a different invoker than the *split* action. It is worth noticing, however, that OpenWhisks offers additional features that we did not consider, such as automatic scaling of service instances, and integration with other services, e.g., databases and push notifications.

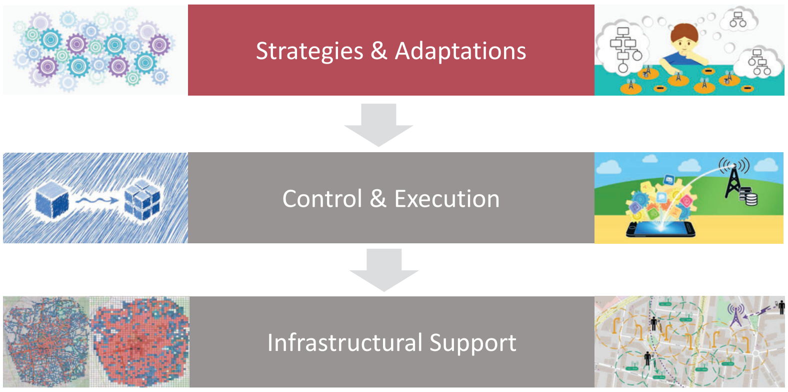
7.7 Conclusion and Outlook

In this chapter, we presented *flexEdge*, a flexible framework for the execution of microservices at the edge. Most notably, we introduced the concept of *store-based onloading* to avoid expensive transferring or pre-provisioning of offloadable code. Instead, microservices are fetched from a *microservice store* and running service instances can be shared across clients. We demonstrated the benefits that this approach has with regards to end-to-end latency and energy consumption on the client device. Using our approach, we could achieve a maximum reduction in energy consumption of 94 % and up to 13 × lower end-to-end latency. This maximum benefit was measured for a microservice that performs object detection, representing a realistic use case for Edge Computing (because of its computational complexity and possible usage in many recognition applications). In addition, providing microservices through a store has other advantages that were not considered in detail and which we leave for future work, such as reusability. For smaller (in terms of their size) microservices the benefit of our approach was reduced drastically, leading to the conclusion that—similar to current practices in offloading frameworks—onloading also requires a careful tradeoff w.r.t. which services to execute outside the client device. We furthermore demonstrated that our approach of using distributed message queues to realize function chaining outperforms a popular platform for Serverless Computing.

Although flexEdge provides the basis for executing microservices at the edge, we only touched briefly on a number of decisions that need to be taken at the controller. Some of those decisions will be addressed later in Part IV of this thesis. For instance, Chapter 8 will present a placement strategy for functional parts of applications. This strategy can be implemented for the placement of microservices into the framework that we have proposed in this section. Chapter 10 shows how microservices can be adapted at runtime by providing different service variants. Future work should investigate instance lifetime management in more detail, especially the tradeoff between cold start latencies and resource utilization. Furthermore, if we assume multiple instances of a microservice are running, efficient strategies for the user-to-instance assignment are required.

Part IV

Strategies & Adaptations



The preceding Part III introduced an Edge Computing framework based on the concept of composable microservices. In this part, we propose strategies and adaptations that can be implemented into such an Edge Computing framework. Such mechanisms are invoked, e.g., when services are requested or changes in the execution environment require actions such as the re-placement of services.

Two chapters (Chapter 8 and Chapter 9) are concerned with *placement strategies*, while Chapter 10 *adapts* the microservices of our Edge Computing framework at runtime.

Chapter 8 presents a heuristic approach for the *operator placement problem*, i.e., where to place functional parts of an application. Chapter 9 extends the placement problem to *data* captured at the edge and proposes a novel approach for providing *micro-storage capabilities at the edge* by taking into account the user's context.

While Chapters 8 and 9 make planning decisions, Chapter 10 adds the dimension of *runtime adaptations* to our Edge Computing framework. To realize this, we re-model our previously introduced concept of microservices, adding *service variants* and making them adaptable at runtime, e.g., to trade runtime for computation quality.

CHAPTER 8

Operator Placement[1]

Chapter Outline

8.1 Introduction

In the previous Part III, we have introduced a control and execution framework for Edge Computing. This framework provides the basis to manage the lifecycle of edge deployments. Most notably for this section, the framework has to make the decision on where to place (connected) components of an application, given a set of networked nodes that make a certain amount of resources available. This section will be focused on precisely that decision.

Placing topologies of processing units onto network topologies has been widely researched, e.g., in the context of complex event processing (CEP), and labeled *operator placement*. Operators in a broad context are functional components, often lightweight, that carry out a specific task. Operators can be further characterized by certain properties, such as their resource requirements and required bandwidth (e.g., for forwarding the result of a computation). In the majority of cases, operators do not stand alone, but are part of an *operator graph*, i.e., a chain of multiple

[1] Large parts of this chapter are verbatim copies from [Ged+18e]. Those text segments are printed in *gray color*. Tables and figures taken or adapted from this publication are marked with † in their caption.

operators. The edges in such a graph represent the logical flow of data between operators for subsequent computing steps.

For these reasons, the lightweight microservices of our Edge Computing framework as defined in Section 7.3.1, can be considered as one example of operators. Another example would be function-as-a-service instances (see Section 7.2.3) or application-level remote application procedure calls (RPCs). In this section, we will therefore examine the placement of microservices in the terminology of the established problem domain of operator placement. While this includes some restrictions on our model, e.g., w.r.t. the topology of operator graphs (see Section 8.2 and Section 8.3.2) the placement strategies presented in this chapter can be applied to a number of applications in the domain of Edge Computing, e.g., in streaming analytics where data originates from sensors at the edge.

The motivation for this chapter stems from the fact that solving large instances of the operator placement problem optimally is computationally hard [EL16] and, thus, unfeasible in practice. However, in many cases, placement decisions have to be made quickly. For instance, offloading computations often happens on-demand, when a certain service is requested. Delays in the placement decision slow down the overall provisioning process for that service, which can lead to unsatisfactory experiences for users. Being able to make placement decisions quickly also allows for frequent reconfiguration. For instance, this is required in case of user mobility or changes in service demands. Therefore, reducing the time it takes to compute an assignment of operators to network nodes is crucial in large-scale dynamic environments.

While the general operator placement problem has been extensively researched, some of the existing works consider homogeneous environments, e.g., the placement of operators or network functions in data centers, and focus on the optimization of either resource consumption of computing nodes or on network metrics. However, we have seen that this does not represent an Edge Computing environment, where available resources and network conditions are heterogeneous in various aspects. When mapped to such a heterogeneous environment, the placement problem becomes more challenging.

Overview of approach. Following the discussion from Section 2.1, we want to include all resources on the continuum between end users and the cloud in our model. To model the difference between very proximate resources and those further in the core network, we introduce an architecture as a reference model that consists of three tiers: edge, fog, and cloud nodes. The nodes typically differ across the tiers in terms of placement cost, link quality, and computational capacity. Because these tiers help us to capture various (edge) computing devices, we summarize the problem this chapter addresses with the general term *in-network operator placement problem* (INOP).

We propose a heuristic approach that works by modifying the input to an ILP model that represents the INOP problem. More specifically, we introduce constraints on the placement decisions that reduce the solution space, and hence, the solving time. These constraints exploit the characteristics of the individual tiers of our edge-fog-cloud architecture and the topologies of both the network and the operator graphs. To this end, we define three general heuristic approaches: (i) restricting the placement of a certain operator to a subset of nodes, (ii) fixing the placement of an operator to exactly one node (pinning), and (iii) enforcing the colocation of operators on the same nodes. For each of our heuristic approaches, we implement example instances to demonstrate the feasibility of this approach. Besides the individual heuristics, we also investigate their combination. We show that this approach considerably reduces the time required to compute a placement decision while only leading to a small optimality gap.

Summary of contributions. In summary, this chapter provides three main contributions:

- We define a model for the heterogeneous in-network operator placement problem (Section 8.3). We formulate this problem in the context of a 3-tier architecture consisting of edge, fog, and cloud nodes.
- We propose an approach for reducing the solving time of the placement problem by using heuristics that modify the original placement problem. Section 8.4 introduces three general classes of such heuristics: (i) placement restriction, (ii) operator pinning, and (iii) operator colocation.
- For each of these classes, we implement representative instances in a testbed (Section 8.5). Furthermore, besides applying the heuristics individually, we suggest three combinations of them. We evaluate our approach with respect to the reduction in the solving time of the problem and the introduced optimality gap (Section 8.6).

8.2 Related Work

The problem of operator placement has been studied extensively, especially in the context of (distributed) stream processing [Che+03; Ste97] and (distributed) complex event processing (CEP) [CM12; CM13]. Lakshmanan et al. [LLS08] survey and classify placement strategies for stream processing systems. The authors present a general definition of the placement problem, in which operators in a logical flow graph are assigned to processing nodes in a physical topology. Besides operators, a logical flow graph contains data sources (termed *producers*) and sinks (termed *consumers*). Our model (see Section 8.3) shares these basic characteristics in its definitions of operator graphs and an underlay network. In addition, the survey provides an analysis regarding which of the surveyed approaches is applicable in which domain. Hirzel et al. [Hir+13] survey different optimization mechanisms for stream processing. Among the optimization mechanisms, placement is defined as assigning operators to hosts and cores. An important observation of the survey is that placement trades communication costs for resource utilization when several operators are placed on the same host. In the colocation heuristic we will introduce in Section 8.4.3, we will consider this observation to enforce the colocation of certain operators.

Network-awareness. We can distinguish between *network-agnostic* [Che+03; Pen+15] and *network-aware* approaches [Pie+06; RDR10; ZA08]. The former do not consider network characteristics (e.g., the latency and available bandwidth on the links), while the latter do. Considering network characteristics is crucial in the Edge Computing scenarios we examine because delays have a considerable impact on the delivered quality-of-service. It therefore makes sense to consider both the network and the resource dimension (i.e., how many resources are available at specific nodes and what the costs of placing functionality on the nodes are). Our proposed model considers both the network and the resource dimension in the operator placement problem.

Pietzuch [Pie+06] and Rizou [RDR10] present approaches to minimize the network usage. However, they do not consider the differences in placement costs or constraints such as available bandwidth on the network links. Carabelli et al. [Car+12] present a linear programming relaxation for a placement problem that minimizes the bandwidth-delay product.

Homogeneous vs. heterogeneous environments. Similar to our work, Cardellini et al. [Car+16] provide a comprehensive model for the operator placement problem. However, in their scalability analysis of the problem, only a homogeneous environment is considered. Nardelli et al. [Nar+19] consider a heterogeneous environment to place components of a stream processing system. However, the authors do not model the different locations of data sources and sinks along the edge–cloud continuum. Similarly, Sharma et al. [Sha+11] investigate the task placement problem in an heterogeneous environment that is limited to Cloud Computing infrastructure.

Unrealistic assumptions for Edge Computing environments. Several works such as [ZA08] and [Hua+11] do not allow the placement of multiple operators on one node, or consider an equal number of operators and nodes [MK16]. We argue that these are unrealistic assumptions for Edge Computing scenarios. Especially the first is in stark contrast to multi-tenant Edge Computing environments (see Section 3.5.2). Others assume uniform capacity of the processing nodes [EL16] or restrict the underlying topology, e.g., to a tree topology [SP15]. In contrast, our heuristics do not have these restrictions.

Comparable approaches. User-defined constraints for the placement are introduced in [TLL14], but no network costs are taken into account. Furthermore, the constraints have to be specified manually by the user. In contrast, our approach automatically generates placement constraints from a given problem input and the characteristics of our 3-tiers of edge, fog, and cloud nodes. Closest to ours is the work of Bahreini and Grosu [BG17]. The authors propose a heuristic approach to the placement problem of multi-component edge applications. Contrary to our problem formulation, they take into account user mobility but do not model three tiers of data source and link locations—something we argue is crucial to capture the real-world characteristics of edge applications.

Comparison to similar problems. The problem of operator placement has some similarities to the virtual network embedding (VNE) problem, e.g., the placement of virtual network functions [Coh+15]. Similarly, Even, Rost, and Schmid [ERS16] investigate the problem of placing SDN functions. However, virtual network embedding mostly deals with the placement of virtual networks or virtual network functions in data center environments [Wan+15]. Compared to the problem we address in this chapter, the infrastructure on which the network functions are placed is more homogeneous w.r.t. the resources they offer. In contrast to that, we address

TABLE 8.1: NOTATION OF THE INOP PROBLEM

V	Set of underlay nodes
SRC	Set of data sources
SNK	Set of data sinks
E	Set of underlay edges
$G = (V \cup SRC \cup SNK, E)$	Underlay graph
C_v	Node capacity
B_e	Link bandwidth
O	Set of operators
w_o	Operator workload
F	Set of edges in the operator graph
f_{o_1,o_2}	Bandwidth requirement of data flow between operators
$H = (O \cup SRC \cup SNK, F)$	Operator graph
$x_{o,v}$	Binary decision variable for operator placement
$y^{u,v}_{o_1,o_2}$	Binary decision variable for routing of operator flows
α	Cost weight factors
$p_{o,v}$	Placement cost
P	Aggregated placement costs
$q_{u,v}$	Link cost
Q	Total link cost

the heterogeneity of nodes on the different tiers (edge, fog, and cloud). Furthermore, existing work on VNF placement considers only one data source [Lui+15], whereas our operator graphs can have multiple data sources.

In some peer-to-peer networks, dedicated nodes, referred to as *brokers* exist. Brokers are intermediaries and relay information between regular nodes in the network (e.g., to implement discovery mechanisms). Placing such brokers is often done w.r.t. to the latency when the given brokers are overloaded [GES08].

Summary. In conclusion, related work in the domain of operator placement comes mostly from the domains of (distributed) stream processing and (distributed) complex event processing. It is worth noting that—same as our model—works from this domain assume a restricted topology of the operator graph(s), in the sense that they are acyclic. While we adopt these limitations, our approach differs in the sense that we operate on a network of nodes in a 3-tier architecture of edge, fog, and cloud nodes. Based on this architecture, we define placement constraints in a heterogeneous (w.r.t. resources and connection characteristics) network topology.

8.3 System Model and Problem Formulation

In this section, we describe the formal model for the in-network operator placement (INOP) problem. Based on the introduced model, we formulate the placement problem with an integer linear program (ILP). A summary of the notation can be found in Table 8.1.

8.3.1 Underlay Network

We define the underlay network as the joint representation of networked nodes on which computations can be carried out, the producers and consumers of data (hereinafter termed *sources* and *sinks*), and the physical interconnections between them. This definition is in line with common models for the placement problem in stream processing systems (see [LLS08]). Hence, the underlay models our infrastructure for computation and communication. We model the underlay network as an undirected graph $G = (V \cup SRC \cup SNK, E)$, where V is the set of nodes that can host operators. In addition, there are two special types of nodes that represent sources that emit data to be processed (SRC) and sinks where the result of the processing is to be delivered (SNK). These nodes can for example represent sensors, or actuators at the start and end of an execution pipeline, following the model of CEP and stream processing. $E = \{\langle u, v\rangle \mid_{u,v \in V \cup SRC \cup SNK}\}$ denotes the set of links interconnecting the nodes.

Each node $v \in V$ has a capacity C_v, which denotes the maximum computational load that node v can handle. We can easily model the case where the node is not part of the computing infrastructure (for example, where it just represents a forwarding middlebox) by setting $C_v = 0$ for that particular node. For each link $e = \langle u, v\rangle \in E$, we consider that its capacity is upper-bounded by its bandwidth B_e. Note that multiple data streams (e.g., from different operator graphs, see next Section 8.3.2) may be routed through one underlay link, and therefore, the upper bound applies to the *aggregated* data flows on that link. We assume that routing in the network is according to shortest paths, which is in line with current data communication networks.

8.3.2 Operator Graphs

In our in-network processing scenario, data is generated by the sources nodes, processed by a well-defined series of operators and ultimately transferred to data sinks. Operator graphs are linear sequences and can have multiple source and sink nodes. We model this logical flow of data as an *operator graph*, which is a directed acyclic graph (DAG) $H = (O \cup SRC \cup SNK, F)$, where O is the set of operators, and SRC and SNK the same set of data sources and sinks as defined in Section 8.3.1. $F \subseteq \{\langle o_1, o_2\rangle \mid_{o_1,o_2 \in O \cup SRC \cup SNK}\}$ represents the data flows between the operators, sources and sinks. Note that H can also be used to represent multiple operator graphs by including them as edge-disjoint subgraphs into H. Each operator $o \in O$ is characterized by a workload w_o, which represents the computational capacity that is required to execute the operator. For each flow $\langle o_1, o_2\rangle \in F$, f_{o_1,o_2} denotes the average bandwidth requirement for the transfer of data between nodes in the operator graph. For the placement, we assume that there is at least one link from each source node that satisfies this source's outgoing bandwidth demand, i.e.,

$$\forall src \in SRC \;\; \exists (src, o) \in E : B_e \geq f_{src,o} \tag{8.1}$$

8.3.3 Operator Placement

The INOP problem aims to make decisions on the placement of operators from an operator graph on the underlay nodes, i.e., the actual computing resources. To capture the structure of this decision-making, we use binary decision variables as follows. We introduce $x_{o,v} \in \{0, 1\}$, which characterizes the placement decision of each operator $o \in O$ to every underlay node $v \in V$. We set $x_{o,v} = 1$ if operator o is placed on node v; otherwise $x_{o,v} = 0$. Every placement decision is unique and operator workload cannot be fragmented, i.e., an operator can only be placed on exactly one node. We model this through the following constraint:

$$\sum_{v \in V} x_{o,v} = 1, \forall o \in O. \tag{8.2}$$

In addition, operators have to be placed subject to node capacity constraints, i.e.,

$$\sum_{o \in O} x_{o,v} w_o \leq C_v, \forall v \in V. \tag{8.3}$$

For a pair of operators $o_1 \in O$ and $o_2 \in O$, where $\langle o_1, o_2 \rangle \in F$ and each underlay link $\langle u, v \rangle \in E$, we introduce indicating variables $y^{u,v}_{o_1,o_2} \in \{0, 1\}$ to represent whether flow $\langle o_1, o_2 \rangle$ will be routed through underlay link $\langle u, v \rangle$. Note that in practice, many instances of this variable are zero, because it has to be set for every possible combination of flows between operators and underlay links. However, modeling the problem in such a way is practical for ILP solvers in order to aggregate the consumed bandwidth on an underlay link. The bandwidth constraints for the underlay links should not be violated, i.e.,

$$\sum_{\langle o_1, o_2 \rangle \in F} y^{u,v}_{o_1,o_2} f_{o_1,o_2} \leq B_{u,v}, \forall \langle u, v \rangle \in E. \tag{8.4}$$

As we adopt shortest-path-based routing in the underlay network, $y^{u,v}_{o_1,o_2}$ actually depends on both $x_{o_1,v}$ and $x_{o_2,v}$. More specifically, for all $\langle o_1, o_2 \rangle \in F$, we have

$$x_{o_1,u} = \sum_{v \in V} y^{u,v}_{o_1,o_2} \text{ and } x_{o_2,v} = \sum_{u \in V} y^{u,v}_{o_1,o_2}. \tag{8.5}$$

8.3.4 Cost Model

The objective of the INOP problem is to make placement decisions to achieve cost-effectiveness. We consider two types of costs, namely, placement costs and link costs. Because we want to address the placement problem in the context of different stakeholders (see Section 3.4), we assume heterogeneous costs for placing operators on nodes. Besides representing different underlying business models of the stakeholders, the heterogeneous placement costs can also model varying technical efforts that are required to execute a certain operator. This is also dependent on both the operator and the underlay node. Hence, for a given operator $o \in O$ and a given node $v \in V$, the placement cost is given by $p_{o,v}$. With this definition, the aggregated placement cost is given by

$$P = \sum_{o \in O} \sum_{v \in V} p_{o,v} x_{o,v}. \tag{8.6}$$

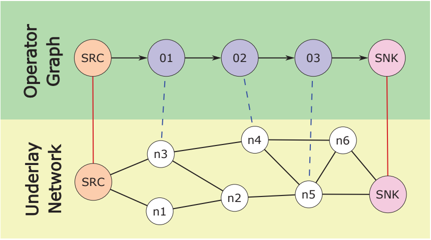
For each link $\langle u, v \rangle \in E$ in the network, a cost $q_{u,v}$ is associated with that link. This cost attribute can be used to represent quality-of-service (QoS) attributes such as latency or the monetary cost for using the link. These link costs occur whenever data flows on the (logical) edges of the operator graph use a physical link in the underlay. The total link costs are therefore given by

$$Q = \sum_{\langle u,v\rangle \in E} \sum_{\langle o_1,o_2\rangle \in F} q_{u,v} \cdot y^{u,v}_{o_1,o_2}. \tag{8.7}$$

We believe that the above two cost types are both practical and generic enough to capture a wide range of real-world performance metrics. To model tradeoffs between the two costs, we introduce a parameter $\alpha \in [0,1]$ to weigh the different costs. Given these definitions, the total cost for a placement decision is given by

$$Cost = \alpha P + (1-\alpha)Q. \tag{8.8}$$

EXAMPLE 8.1: PLACEMENT EXAMPLE



This figure shows a simple instance of the placement problem. Source-sink pinnings are shown by the solid red lines. Blue dashed lines represent a possible result as returned by a placement algorithm. In our evaluation, we will consider a more complex underlay network as well as different types of operator graphs.

8.3.5 Problem Formulation

Based on the presented model, we formulate the INOP problem as follows: For a given underlay network and operator graphs[2] find an assignment for each operator to an underlay node, such that the cost function (8.8) is minimal. An example of an outcome for a placement decision is shown in Example 8.1. More formally, we formulate the problem with the following integer linear program (ILP).

$$\begin{array}{ll} \text{min} & Cost \\ \text{s.t.} & (8.2), (8.3), (8.4) \\ & x_{o,v} \in \{0,1\}, \; \forall o \in O, \; \forall v \in V. \end{array}$$

This ILP model is the basis for our evaluation of placement approaches. Section 8.5 will detail how we expressed the INOP problem in a testbed implementation using a modeling language for optimization problems.

8.3.6 Edge-Fog-Cloud Architecture

Contrary to previous works in this domain, we examine the placement problem specifically in the context of in-network processing, where we consider the network to be in a 3-tier architecture, consisting of edge, fog and cloud nodes. This distinction follows the (sometimes subtle) distinction between edge and fog that is

[2] Recall that our model of operator graphs as described in Section 8.3.2 allows to represent multiple operator graphs using the set notation of one DAG.

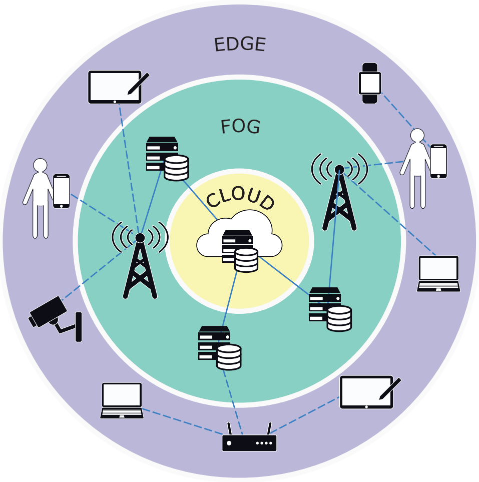


FIGURE 8.1: 3-TIER ARCHITECTURE OF EDGE, FOG, AND CLOUD NODES

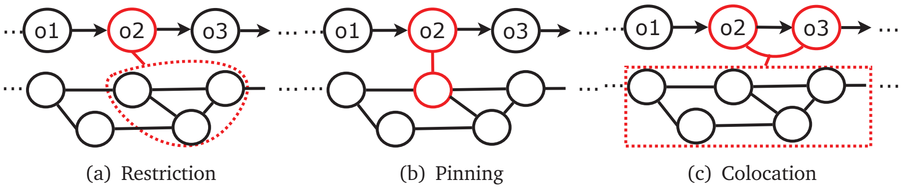
commonly found in literature (see Section 2.1). It is important to note that the tiers include computing nodes as well as data sources and sinks, since those can also be located in different tiers. Cloud nodes represent data center infrastructures while fog nodes are middleboxes or gateways for end devices. Edge nodes are the end devices or devices in immediate proximity of the user. These different types of nodes allow us to model the different characteristics of the devices encountered in the respective tiers, e.g., with regards to their capacity and network connectivity. For our placement heuristics, we can further leverage the specifics of these 3 tiers. Note that contrary to, e.g., the definitions of MEC[3], we model powerful resources at gateways to be in the fog tier. As an example, depending on the locations of the sources and the sinks, we can restrict the placement of operators to a certain subset of underlay nodes, such as underlay nodes in a certain tier. Figure 8.1 shows the 3-tier architecture with example devices in each tier.

> EXPLANATION 8.1: NETWORK TOPOLOGY
>
> We assume cloud nodes to be fully connected, fog nodes to resemble a LAN/WAN topology, and edge nodes to have a connection to at least one fog node that acts as a gateway for this edge node. Edge nodes are organized in clusters, depending on which fog node they are connected to. Furthermore, edge nodes have a probability to be connected to more than one fog node and an ad-hoc connection probability to other edge nodes.

8.4 Heuristic Approach

To reduce the solving time of the placement problem, i.e., the time it takes to compute a placement, we propose an approach in which we modify the original problem input. We use an ILP solver to compute both the optimal solution and a solution

[3] Mobile Edge Computing, see Section 2.1



(a) Restriction (b) Pinning (c) Colocation

FIGURE 8.2: HEURISTICS FOR OPERATOR PLACEMENT†

that uses our heuristics. For the optimal solution, the input file for the solver contains only the definition of the underlay and operator graphs. In addition, we are also given a matrix of the placement costs, depending on the operator and underlay node. Our heuristics modify the original input problem for the solver in such a way that additional constraints on valid placements are added. More specifically, we impose a set of extra constraints on the placement decision variables $x_{o,v}$. As demonstrated in other publications [TLL14], adding constraints can lower the time necessary to compute a solution. Note that we use the term *heuristics* in a general sense to denote a practical approach to simplify a complex problem. Our heuristics do *not* change the computational complexity of the INOP problem but—as will be shown in Section 8.6.2—lead to considerably reduced solving times, making our approach applicable in practice.

In the following, we propose three general classes of heuristics that follow this approach: (i) placement restriction, (ii) operator pinning, and (iii) operator colocation. All three impose constraints on the placement decisions that allow to naturally model properties of our 3-tier model as presented in Section 8.3.6. For example, colocation can reduce expensive bandwidth transfers between nodes that are located in different tiers, while restriction and pinning can be used to enforce non-functional requirements, such as keeping one's computations on trusted nodes only.

8.4.1 Placement Restriction

Placement restrictions limit the placement of an operator to a subset of nodes (see Figure 8.2(a)), i.e., for each $o \in O$, we define a subset $V' \subset V$ and we set

$$\sum_{v' \in V'} x_{o,v} = 1. \tag{8.9}$$

This heuristic can be used to enforce the placement on nodes with desired properties, such as low placement cost, good link connections, or proximity to the sources and the sinks. In real-world deployments, one might also want to restrict certain operations to a specific geographic region, e.g., because of privacy considerations. We implement this heuristic as follows: for each connected sub-graph of the operator graph H, we determine the locations of the sources and the sinks, i.e., on which network tier they reside. If at least one of the sources or sinks are located in the cloud tier, we restrict the placement of the operators in the graph to the cloud and fog nodes only. However, if either a source or sink is located at the edge of the network, we try to avoid expensive cloud links and restrict the placement of all the operators to either the edge or fog tier. It is important to note that we do not consider fog nodes to be either the source or sink of data, since we consider them to be network middleboxes or gateway nodes that do not produce or consume application data. Our placement restriction approach can be formalized as follows:

$$\begin{array}{ll} \min & Cost \\ \text{s.t.} & (8.2), (8.3), (8.4), (8.9) \\ & x_{o,v} \in \{0,1\},\ \forall o \in O,\ \forall v \in V,\ \forall v' \in V', \\ & V \cap V' = \emptyset. \end{array}$$

8.4.2 Operator Pinning

Operator pinning is the most restrictive heuristic. It enforces the placement of an operator to one particular node as depicted in Figure 8.2(b). Therefore, it can be considered a special case of the more general restriction heuristic described in Section 8.4.1. However, the rationales we use for determining the restriction and pinning are different. Restrictions operate based on the locations of the data sources and sinks in the different tiers, while for the pinning, we try to find a suitable underlay node in proximity of sources and sinks, irrespective of their locations in the tiers. Therefore, we treat restriction and pinning as different types of heuristics. According to our model, for a pinned operator-node pair $(v \in V, o \in O)$, we set $x_{o,v} = 1$ to represent the restrictions introduced by this heuristic.

For the operator pinning heuristic, we implement the following logic: for each operator graph, we try to pin the first operator, i.e., the one adjacent to one or multiple source nodes. For the candidate nodes to place this operator, we have to distinguish between two cases: (i) a single source and (ii) multiple sources connected to the operator. In the first case, candidate nodes are the source node and its one-hop neighbors in G. In the second case, we consider the 2-hop neighborhoods of each of the sources. Out of these subgraphs, we define the candidate nodes as all nodes that appear in all these 2-hop neighborhoods. Then, for each candidate node v, we compute a penalty function

$$s = \alpha p_{o,v} + (1-\alpha)\bar{q}_v, \tag{8.10}$$

where o is the operator to be placed, $p_{o,v}$ is the placement cost and $\bar{q}_v$ is the average link cost of the edges that are adjacent to v. We then pin the operator to the node with the lowest penalty value, since this node will have the lowest weighted placement and link costs to host this operator. Consequently, we obtain a set $P = \{(o_1, v_1), \dots (o_n, v_n)\}$ of all operator-node pinnings and it must hold that

$$\forall (o_i, v_i) \in P : x_{o_i, v_i} = 1. \tag{8.11}$$

Applied to our original model, pinning results in the following optimization problem:

$$\begin{array}{ll} \min & Cost \\ \text{s.t.} & (8.2), (8.3), (8.4), (8.11) \\ & x_{o,v} \in \{0,1\},\ \forall o \in O,\ \forall v \in V,\ P \cap V = \emptyset. \end{array}$$

8.4.3 Operator Colocation

While our model naturally allows for operators to be colocated on one node, this heuristic enforces the colocation of certain operators to one underlay node. Formally, we define a pair of operators ($o_1 \in O, o_2 \in O$), and it must hold true that $\exists v \in V,\ x_{o_1,v} = x_{o_2,v} = 1$. As an example, Figure 8.2(c) depicts the colocation of operators o_2 and o_3. The colocation of operators requires a tradeoff between placement costs and communication costs. Consider a scenario where operator o_1 precedes operator o_2. Colocating these two operators on one underlay node is most likely beneficial to the overall utility of the system if operator o_2 is lightweight in terms of resource utilization and placement costs, and the communication between o_1 and o_2 requires high bandwidth. To model this, we compute a score for each pair of neighboring operators o_1 and o_2, i.e., $\langle o_1, o_2 \rangle \in F$, defined by

$$s = \frac{f_{o_1,o_2}}{\alpha(\bar{p}_{o_1,v} + \bar{p}_{o_2,v} + w_{o_1} + w_{o_2})}, \tag{8.12}$$

where f_{o_1,o_2} is the bandwidth of the flow between the two operators, $\bar{p}_{o,v}$ denotes the average placement cost of the operator (across all underlay nodes) and w_o the workload of the operator. Since colocation does not restrict the set of nodes on which the colocated operators can be placed, we choose the sum of the average placement cost across all underlay nodes to represent the impact of the costs. Note that this can be replaced by other metrics, such as the median value or a certain percentile. The score will favor the colocation of operators with high bandwidth demands, while taking into account their placement costs and workload. We then select n_o operator pairs with the highest score to colocate. Empirically, we determined $n_o = \lfloor |O|/5 \rfloor$ to be an appropriate value. Mapped to our optimization problem, we then get a set $O_C = \{(o_1, o_2), \dots (o_n, o_{n+1})\}$ for which the following must hold:

$$\forall (o_i, o_j) \in O_C : \exists v \in V, x_{o_i,v} = x_{o_j,v} = 1 \tag{8.13}$$

Our modified problem for the colocation of operators can therefore be defined as follows:

$$\begin{array}{ll} \min & Cost \\ \text{s.t.} & (8.2), (8.3), (8.4), (8.13) \\ & x_{o,v} \in \{0,1\},\ \forall o \in O,\ \forall v \in V,\ O_C \cap V = \emptyset. \end{array}$$

8.4.4 Strategies for Combining Heuristics

While the three heuristics described above can be applied individually, we can think of ways to combine them, i.e., to apply more than one heuristic for a given input problem. Figure 8.3 shows a flowchart that describes the different possible ways of combining our heuristics. We first have the option to pin certain operators to an underlay node. Then, the more general, i.e., less restrictive, heuristics *restriction* and *colocation* can be applied independently. It is important to note that these combinations do not work in isolation but influence each other and might be contradictory. This is especially true for the pinning heuristic. We consider the following three combinations of heuristics:

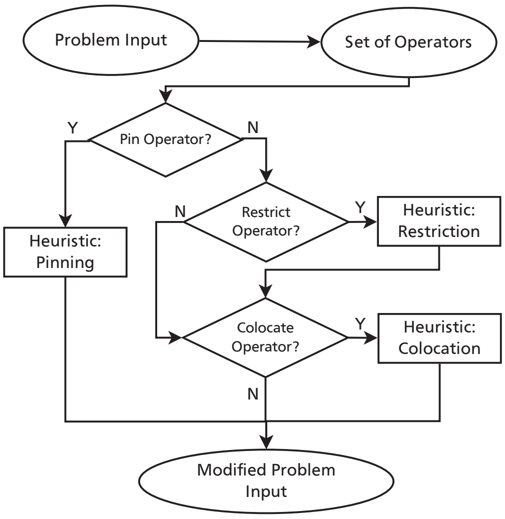


FIGURE 8.3: FLOWCHART DENOTING THE SEQUENCE OF PLACEMENT HEURISTICS[†]

COMB-1: PINNING → RESTRICTION | As the first combination, we apply pinning and then restriction of operators. Note that the restrictions will not be applied to already pinned operators, because this might lead to contradictions, e.g., the pinning computes the placement of an operator to a node which is not in the set of possible nodes according to the restriction heuristic.

COMB-2: PINNING → COLOCATION | For this combination, we first apply pinning and additionally the colocation of operators. Similar to COMB-1, operators that are pinned will not be in the candidate set considered for colocation. For example, consider that the pinning heuristic fixes the placement of an operator o_1 on a node A, and the colocation heuristic enforces the colocation of operators o_1 and o_2, but the capacity of node A is less than the workload of o_1 and o_2 combined. Therefore, when pinning is applied before other heuristics, we do not consider the pinned operators for other heuristics applied afterward.

COMB-3: RESTRICTION ⇄ COLOCATION | Lastly, we leave out the pinning heuristic and combine restriction with colocation. Contrary to the previous combinations, this does not require the exclusion of operators from the heuristic that is applied second and, hence, the two heuristics can be applied in arbitrary order.

8.5 Testbed Implementation

To evaluate our placement heuristics, we built a simulation-based testbed. Its main components are implemented in Python. Figure 8.4 shows the components of the testbed. The main component that carries out the simulations (`simulation.py`) takes a configuration file as input. This file specifies the simulation parameters, such as the sizes or properties of input graphs. Accordingly, a graph generator (`graphgen.py`) is responsible for creating underlay and operator graph inputs. Our proposed heuristics are implemented in separate source files (`heuristics_restriction.py`, `heuristics_pinning.py`, and `heuristics_colocation.py`) and the heuristic's functionalities called from the main simulation component.

We used Pyomo[4] as a tool to model the ILP problem in Python. The file `model.py` implements our model as defined in Section 8.3. The complete source code of the model can be found in Appendix F. Another component of our simulation utilities (`solver.py`) is responsible for interacting with the concrete ILP solver. It submits the (modified) problem to the solver and parses the results (i.e., the placement decisions and the time the solver took). Pyomo features support for different solvers. For our evaluation, we used the *GNU Linear Programming Kit*[5] (GLPK) as a solver for the ILP problem.

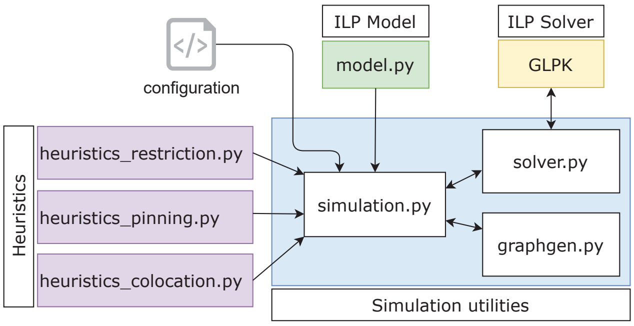


FIGURE 8.4: TESTBED IMPLEMENTATION

8.6 Evaluation

Using our implemented testbed as described in the previous Section 8.5, we now evaluate our approach. We first describe our experimental setup (Section 8.6.1). Our evaluation quantifies the gain in resolution time for the placement (Section 8.6.2) and the optimality gap of the heuristic-based solutions (Section 8.6.3). We further discuss and compare our results in Section 8.6.4.

8.6.1 Experimental Settings

8.6.1.a Underlay network

For our evaluation, we define three network underlays of different sizes. Their characteristics are shown in Table 8.2. To take into account the different characteristics of nodes and connections in the different tiers, Table 8.3 summarizes the values we set for the capacity, link costs, and available bandwidth. The parameters for the underlay networks were chosen such as to represent the differences in the devices' characteristics on each tier. Recall that the link cost property can be used to model different properties related to the usage of that particular link, e.g., the latency.

[4] https://www.pyomo.org/ (accessed: 2019-11-13)
[5] https://www.gnu.org/software/glpk/ (accessed: 2019-11-13)

TABLE 8.2: UNDERLAY NETWORK†

	Underlay 1	Underlay 2	Underlay 3
Cloud nodes	2	5	10
Fog nodes	5	10	20
Edge clusters	2	3	5
Edge nodes	16	36	70
Edge-fog connection probability	0.1, 0.3	0.1, 0.3, 0.2	0.1, 0.3, 0.2, 0.1, 0.4
Edge ad-hoc connection probability	0.1, 0.3	0.1, 0.3, 0.2	0.1, 0.3, 0.2, 0.1, 0.4

TABLE 8.3: PROPERTIES OF NODES†

	Cloud nodes	Fog nodes	Edge nodes
Capacity	100	random(20,60)	random (5,20)
Link cost	$\mathcal{N}(200, 2500)$	$\mathcal{N}(20, 4)$	$\mathcal{N}(5, 1)$
Bandwidth	10 000	random(1000,5000)	random(10,30)

8.6.1.b Operator graphs

To represent different types of applications that are to be deployed in the network, we define different types of operator graphs, as shown in Figure 8.5. They differ in the number of operators (labeled o_i), sources (labeled src), and sinks (labeled snk). In addition, the sources and sinks are located in different tiers of the underlay network. This allows us to capture a variety of application scenarios for in-network processing. In detail, we consider four different types of operator graph topologies:

(i) EDGE ANALYTICS | In edge analytics (see Figures 8.5(a), 8.5(b), 8.5(c), 8.5(i), and 8.5(j)) both data sources and data sinks are located at the edge. These operator graphs can for example model the time-critical analysis of sensor data that is gathered at the edge and then fed back to local actuators, e.g., in IoT scenarios as described in Section 4.2.3.

(ii) BIG DATA ANALYTICS | Contrary to edge analytics, the results of big data analytics (see Figures 8.5(d), 8.5(k), 8.5(l), and 8.5(m)) are transferred to the cloud. The motivation for this is that the cloud offers virtually unlimited storage to archive the results. In addition, longer-running analytic tasks are also typically carried out in the cloud.

(iii) HYBRID SINK LOCATIONS | Figures 8.5(e) and 8.5(f) depict examples of operator graphs where the sinks are both located at the edge and in the cloud. This is therefore a hybrid case between the two preceding types. Mapped to a real-world example, this could model an application where the results of a processing pipeline are required for immediate actuation at the edge and also archived in the cloud.

(iv) HYBRID SOURCE LOCATIONS | Besides hybrid sink locations, the sources of the data could also be located at different tiers, as shown in Figures 8.5(g) and 8.5(h). This type of operator graph models applications that require both data sourced at the edge (e.g., contextual data from the environment) and from the cloud (e.g., retrieved from large databases) for their computations.

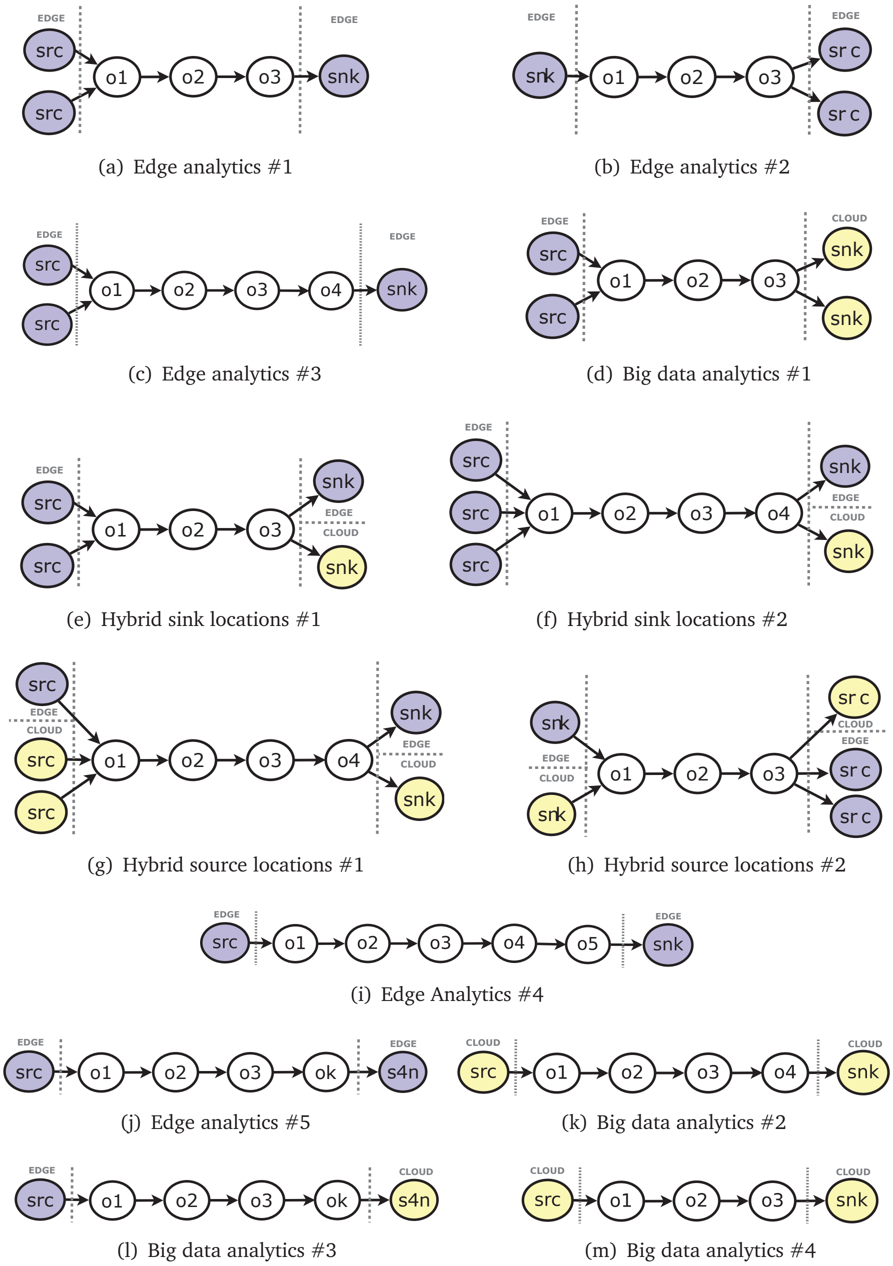
TABLE 8.4: INPUT SIZES FOR THE OPERATOR GRAPHS[†]

	Operator Graphs / Operators	**Sources**	**Sinks**	**Source locations (Edge / Cloud)**	**Sink locations (Edge / (Cloud)**
gs1	5 / 17	8	6	8 / 0	4 / 2
gs2	6 / 20	10	8	10 / 0	4 / 4
gs3	7 / 24	11	9	10 / 1	4 / 5
gs4	8 / 28	14	11	11 / 3	5 / 6
gs5	9 / 32	15	12	12 / 3	6 / 6
gs6	10 / 35	17	15	13 / 4	8 / 7
gs7	11 / 39	19	16	15 / 4	9 / 7
gs8	12 / 44	20	17	16 / 4	10 / 4
gs9	13 / 47	21	19	17 / 4	12 / 4
gs10	14 / 50	22	20	17 / 5	12 / 5
gs11	15 / 54	25	22	20 / 5	13 / 6

From these different types of operator graphs, we generate varying input problem sizes by combining different types of operator graphs. In total, we consider 11 different operator graph inputs, labeled *gs1* through *gs11*. Their properties, such as the number of operators, sources, and sinks are summarized in Table 8.4. Appendix G details which operator graphs are contained in the different inputs. The operators were assigned a random workload between 2 and 5. The required bandwidth between the operators varies between 5 and 20. Placement costs are also randomly generated within the range of 10 to 30. We perform the evaluation using every combination of underlay size and operator graphs described above. For every combination, we report the average results of 5 simulation runs. We set $\alpha = 0.5$, meaning that placement and link costs are weighted equally.

8.6.2 Performance Analysis

First, we analyze the performance of our proposed heuristics, i.e., the reduction in solving time for the placement problem. Figure 8.6 shows the results of this analysis for the different underlay network sizes. The overhead, i.e., the time it takes to compute the placement constraints of the heuristics and apply them to the initial mode, is included in the measurement times reported in this section.



(a) Edge analytics #1

(b) Edge analytics #2

(c) Edge analytics #3

(d) Big data analytics #1

(e) Hybrid sink locations #1

(f) Hybrid sink locations #2

(g) Hybrid source locations #1

(h) Hybrid source locations #2

(i) Edge Analytics #4

(j) Edge analytics #5

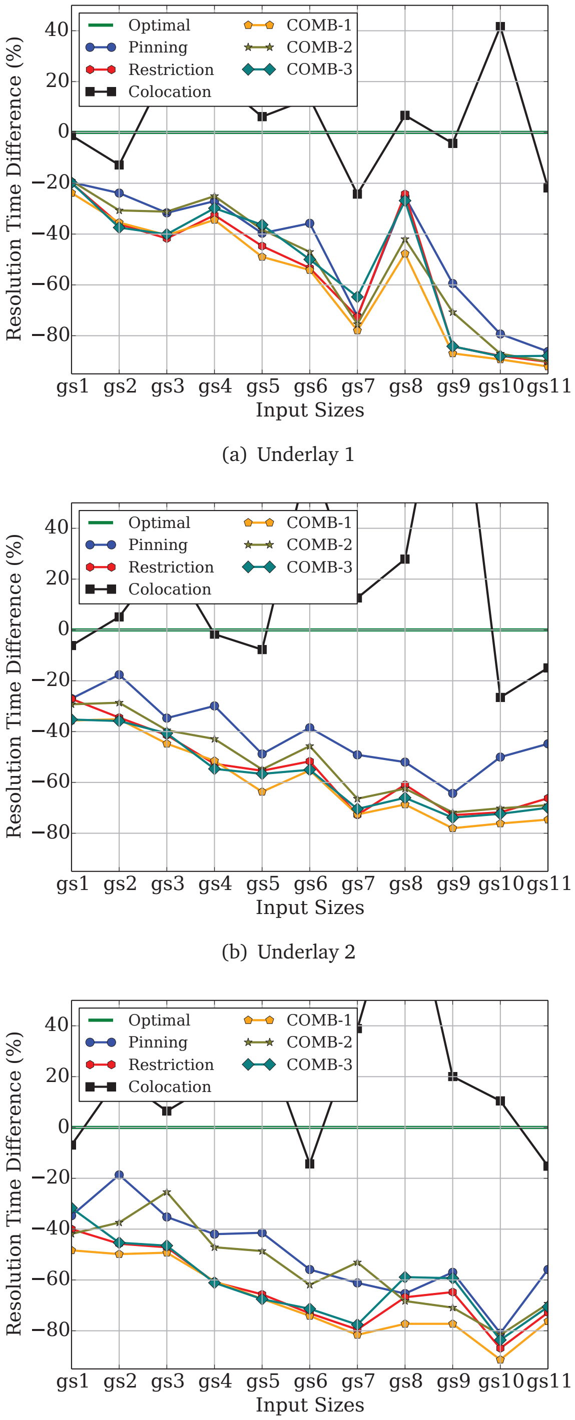
(k) Big data analytics #2

(l) Big data analytics #3

(m) Big data analytics #4

FIGURE 8.5: OPERATOR GRAPH TOPOLOGIES USED IN THE EVALUATION[†]

Except for applying the colocation heuristic alone, all the approaches reduced the resolution time, regardless of the size of the underlay network or operator graph input size. Colocation alone can increase the resolution time because it can make the modified problem more complicated to solve. All our other approaches considerably reduce the resolution time. For the different problem input sizes, we were able to achieve decreases of 20 to about 95 percent on average. It is important to



(a) Underlay 1

(b) Underlay 2

(c) Underlay 3

FIGURE 8.6: EVALUATION RESULTS ON THE RESOLUTION TIME†

emphasize three general observations:

(i) There is a general trend of a larger decrease in resolution time as the size of the problem input increases, meaning that for larger problem instances—as likely in dense Urban Edge Computing environments—we expect our approaches to be even more beneficial.

(ii) We achieve the reduction in resolution time by specifying simple (in the sense that they are easy to determine and naturally consider the characteristics and topologies of Edge Computing environments) placement constraints.

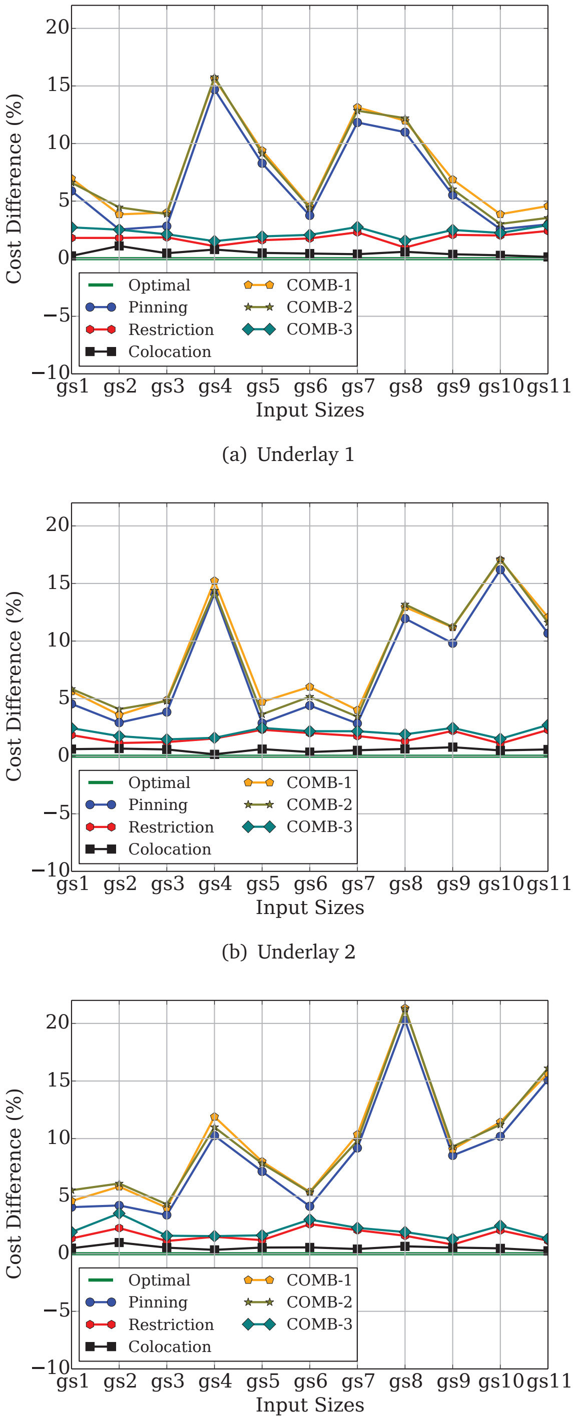
(iii) The savings in resolution time can have a substantial impact in practice, changing unacceptable delays in the provisioning of services to much shorter, acceptable delays. As an example, for one problem instance of *gs7* in *Underlay1*, an optimal solution was computed in 19.15 s, while our best heuristic (in this case the combination of pinning and restriction) led to a solving time of 0.81 s. Included in that time are 0.03 s it took to compute the heuristic and apply it to the original problem input.

Out of the implemented approaches, applying pinning together with restriction was the fastest most of the time. However, we will see in the next subsections that this is also the approach that introduces the largest optimality gap. Pinning alone gives us less benefit for the resolution time because only the first few operators are pinned. Furthermore, fixing the first operator might add complexity for the placement of the remaining operators in the graph. When also applying colocation after pinning, the resolution time is lowered because fewer reasonable placement options are available. In our experiments, restriction and restriction alongside colocation performed similarly w.r.t. the benefit in resolution time.

8.6.3 Optimality Gap

Applying our heuristics will inherently lead to a decrease in the system utility, i.e., the value of the cost function (Equation (8.8)) will increase. However, given the benefits regarding the resolution time, this is a tradeoff one might be willing to accept in practice. Especially in highly dynamic scenarios, reconfiguring placement decisions is time-critical due to quick changes in network characteristics. In addition, other implementations of placement restrictions might represent other placement constraints, e.g., some nodes might be excluded due to privacy concerns. In such cases, the optimal solution might not be applicable in practice at all. In this section, we analyze the optimality gap of our implementation, i.e., we quantify the increase of the cost function for our approaches compared to solving the problem optimally.

In Figure 8.7, we plot this optimality gap. The maximum average increase we can observe is only slightly above 20 % but, in general, it is much lower on average (usually below 5 %). The highest increase is observed whenever pinning is involved since it does not consider the link cost that it might imply for the operators placed later on. The cost difference is the lowest for the colocation heuristic alone because we have more nodes than operators to choose from. Therefore, even though we enforce the colocation of operators, we can find a node that has a combined placement cost close to the optimum. Restriction alone and restriction combined with colocation perform only slightly worse. This is because compared to colocation alone, not all nodes are considered for the placement.



(a) Underlay 1

(b) Underlay 2

(c) Underlay 3

FIGURE 8.7: EVALUATION RESULTS ON THE OPTIMALITY GAP†

8.6.4 Discussion

8.6.4.a Time-cost tradeoff

Based on the results obtained, we now discuss the tradeoff between the saving in resolution time and the cost overhead of our heuristics.

Figure 8.8 depicts this tradeoff with the resolution time on the x-axis and the cost overhead on the y-axis. Each dot represents one result for the different graph sizes. The size of each dot represents the size of the underlay network. Since the colocation heuristic alone often increases the resolution time, sometimes dramatically and therefore does not offer a good tradeoff, we omit it in the plot. From the figure, we can first observe the scalability of our approach since there is a trend towards a bigger saving in resolution time for larger underlay sizes. Second, we can see the tradeoff between the cost optimality gap and saving in time. For instance, COMB-1 leads to good results in terms of time saving but also has one of the highest optimality gaps and a large variance in the optimality gap. Applying restriction alone consistently leads to low cost, however, there is more variance in the saving in resolution time. If we apply restriction and colocation (COMB-3), the results are similar, but for larger underlay sizes, we can see a slight benefit using COMB-3. The similarity of these two is an interesting observation, as we imagine this to be highly dependent on the actual implementation of the colocation heuristic and we plan to examine this in future work. Compared to these, pinning and COMB-2 were found to be the weakest in terms of the tradeoff.

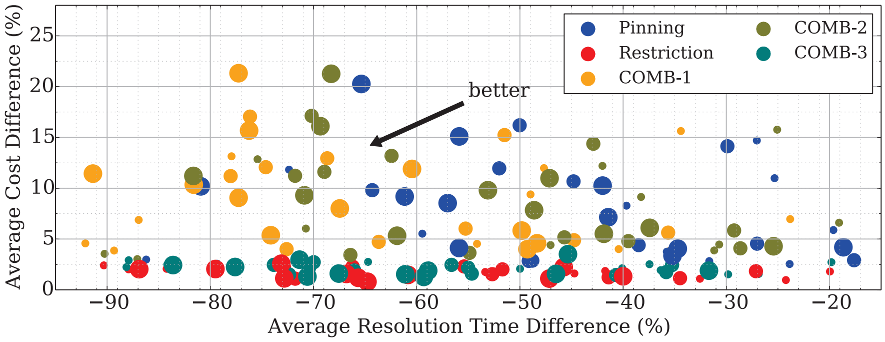


FIGURE 8.8: TIME-COST TRADEOFF FOR THE HEURISTICS[†]

8.6.4.b Comparison with greedy cloud placement

We also compare our results with a greedy algorithm for operator placement. This algorithm places every operator on cloud nodes only and chooses the cloud node with the lowest placement cost for each operator. This is the current practice one would employ to place cloud services without considering the edge or fog as possible tiers for carrying out processing. We plot the results of this greedy placement strategy for the largest underlay network (Underlay 3) in Figure 8.9.

We observe that while the saving in resolution time is comparable to our heuristics (with a maximum saving of around 90 %), the greedy algorithm performs much worse in terms of cost. For larger graph sizes, the costs are three to four times higher than the optimal solution. This is mainly because for the greedy solution, link costs are substantially higher since all data has to be transferred to the cloud, even for operator graphs where data sources and sinks all reside in the edge tier. Compared to that, when applying our heuristics, we saw a maximum increase in the total costs of way below 10 % on average.

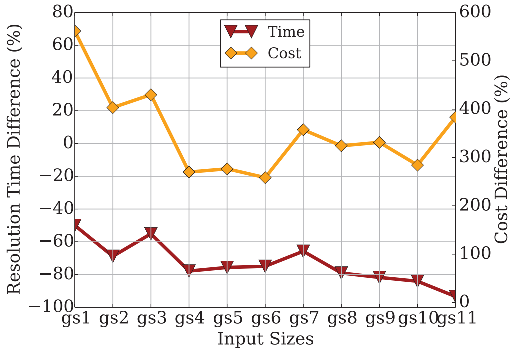


FIGURE 8.9: PERFORMANCE AND OPTIMALITY GAP FOR GREEDY CLOUD PLACEMENTS[†]

8.6.4.c Dissecting the placement decisions

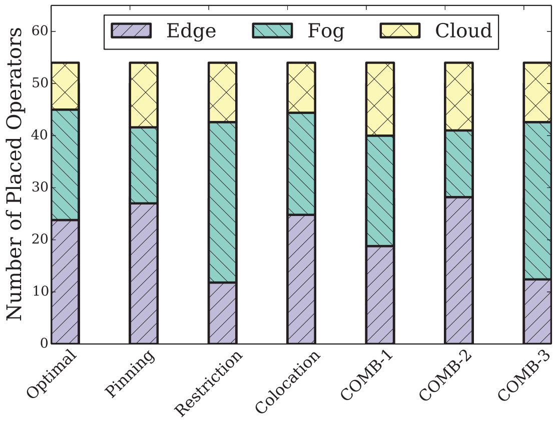


FIGURE 8.10: PLACEMENT LOCATIONS[†]

Figure 8.10 shows how the implemented heuristics influence the placement decisions with respect to the different tiers, i.e., how many operators are placed on the cloud, fog, or edge tier. Recall that for two heuristics—restriction and pinning—a subset of placement decisions will be constrained to a certain tier. The plot serves to illustrates both the differences in how the heuristics function in terms of restricting placement decisions, and how the heuristics affect the non-constrained placements as computed by the solver. We plot the number of operators on different tiers for the largest problem instance (Underlay 3, $gs11$, see Table 8.4 and Table 8.2 for details).

From the results, we can make the following observations: compared to the optimal solution, pinning is more aggressive in terms of placing operators on the edge, i.e., close to where most of our data sources are located. Since restriction always allows the placement on fog nodes (regardless of the location of data sources and sinks), we can clearly see a trend towards fog placement when applying the restriction heuristic either alone or in combination (COMB-1 and COMB-3). In the case of COMB-3, we see nearly the same outcome as with restriction alone. When combining pinning and restriction (COMB-1), we get the highest number of cloud placements. If colocation is applied alone this leads to nearly the same results as the optimal solution. Recall, however, that this increases the resolution time in most cases. With applying colocation after pinning (COMB-1), we see a slight shift towards cloud placements since after pinning, we might have fewer placement possibilities—especially on the fog tier.

8.7 Conclusion and Outlook

This chapter studied the problem of placing operators (i.e., functional part of applications) in a 3-tier in-network topology, consisting of edge, fog, and cloud nodes. We modeled the in-network operator placement (INOP) problem as an ILP model. To reduce the solving time of this computationally hard problem, we presented an approach that modifies the original problem by introducing constraints to the ILP model. First, we introduced three general classes of heuristics to generate such constraints. Then, for each of these classes, we implemented sample representatives to demonstrate their feasibility. By evaluating our approach, we were able to considerably decrease the solving while only introducing a small optimality gap.

Our findings can be applied to a variety of practical problems in the emerging domain of Edge Computing, one of which is the placement of microservices. Hence, this approach can be integrated in the placement decision logic of an Edge Computing framework as presented in Chapter 7.

We envision the following research directions for future work:

Other variants of heuristics | For each of the proposed heuristics, we implemented one sample representative. Future work should explore other variants, e.g., by extending the pinning heuristic to more operators, or defining other metrics for the colocation score.

More general graph topologies | We assumed directed, acyclic graphs in which data sources and sinks were pinned to the underlay network. This follows the model currently found in distributed complex event processing and stream processing applications. Future work should extend this model to more general graph topologies, e.g., such that cycles are allowed. In addition, an extended model should include dynamic re-assignments of source-sink mappings between the underlay network and operator graphs, therefore allowing for a better representation of applications with mobile data source and sinks.

Reducing the computational complexity | Our method was able to considerably reduce the solving time for the INOP problem. However, from the point of view of computational complexity, it remains a hard problem. Future work should examine heuristics that can reduce the general computational complexity of the INOP problem.

CHAPTER 9

Context-Aware Micro-Storage[1]

Chapter Outline

9.1 Introduction

The previous chapter investigated the placement of functional parts of applications (termed *operators*). In this chapter, we extend this placement problem to *data*, i.e., the problem of where information gathered by users at the edge should be stored. Similar to the placement of application components, today most data gathered by end devices is stored in Cloud Computing infrastructures, partly because of the devices' limited capacity [ASH15]. This data flow—from the edge to a cloud location—is opposite to the trend of implementing content delivery networks (CDNs) throughout the Internet. While CDNs aim at providing cache servers for the delivery of content originating from the cloud [Dil+02; VP03], we investigate the opposite data flow from the edge to the cloud. This has been referred to as a *Reverse CDN* [Sch+17; MSM17]. An example of devices that capture data at the edge are mobile phones. Mobile phones nowadays feature a variety of different applications. Data captured by or sent to those applications is usually stored on

[1] Large parts of this chapter are verbatim copies from [Ged+18b]. Those text segments are printed in *gray color*. Tables and figures taken or adapted from this publication are marked with † in their caption. The work was awarded Best Paper at the 2018 *MobiCASE* conference.

distant servers, i.e., in Cloud Computing infrastructures. In any case, the storage location is *inflexible* because it is predefined by the application or cloud infrastructure provider. Furthermore, we see a plethora of different applications that serve the same or similar purpose (e.g., *Dropbox*, *Google Drive*, and *OneDrive* for cloud-based data storage). Although users sometimes use different services to store the same data, the different applications remain isolated from one another and therefore hinder the sharing of data across them. Besides using various applications for the same purpose, the way mobile data is stored and accessed today is completely decoupled from how the data is actually used and what the current usage contexts of users and their intentions are. One example is that users often share content with others at public events, where networks tend to be overloaded. As we will illustrate later, this sharing often happens between users who are present at the same event.

As a result, we are faced with congested networks and high latencies when retrieving data stored at distant locations. Retrieving locally relevant data from distant Cloud Computing infrastructures furthermore incurs big stress on network bandwidth—one of the main motivations for Edge Computing (see Section 3.1). In summary, we can derive the following drawbacks and limitations from the current state of the art:

Drawback I: High retrieval latencies | For data such as video, this has a direct impact on the perceived quality of service and therefore is undesirable. In addition, for many mission-critical applications such as virtual reality or assisted driving, the latencies to cloud infrastructures are prohibitive.

Drawback II: High core network bandwidth utilization | Despite often being retrieved only in a locally restricted area, all data is first sent to the cloud, thus creating high bandwidth utilization and possible bottlenecks in the core network. This is going to worsen as more large-volume data, such as video, will be generated in the future.

Drawback III: Tight coupling and cumbersome sharing | Applications use predefined storage locations, unable to consider where the data will be retrieved. Furthermore, no unified interface is provided for sharing data across different applications and users. While approaches to overcome this drawback could be realized in cloud infrastructures, a novel approach to proximate edge computing creates the opportunity for offering a unified solution that addresses this drawback.

The emergence of Edge Computing provides an opportunity to overcome these drawbacks, by not only providing computing, but also storage capabilities. Unnecessary transfers to the cloud and congestions in transit networks can be avoided if data is stored close to where it actually is retrieved. However, as of today, the questions of how and where to provide storage capabilities for mobile devices at the edge has not been addressed. This chapter closes this gap by presenting a concept termed *vStore* (virtual store).

Overview of concept. vStore is designed as a middleware that abstracts from concrete storage locations and—based on the current usage context and intentions of the user—chooses the most suitable storage location. The decision where to store data is made based on rules that can either be pushed globally to the framework or created individually by users. From a networking point of view, vStore reduces the bandwidth utilization in the core network and the latency when retrieving nearby copies of requested data. From the user perspective, vStore provides context-awareness and facilitates the sharing and reuse of data across locations and applications. Furthermore, by pushing updated storage rules to the client devices, vStore enables network operators and businesses to provide better quality of experience for their customers by providing proximate cloudlet storage and reacting to changes in the network utilization. When making storage decisions, vStore takes the following into account:

- **Type of data**, such as photo, video, contacts, etc.

- **Usage context** as provided by the client. This can include information like time, location, ambient noise level, and network conditions.

- **User intention**, such as private use or sharing of data.

- **Available storage locations** and their properties (e.g. their location in the network).

Instead of solely relying on either local (i.e., on the mobile device itself) or cloud-based storage, we also consider storage locations in the access network or located at the user's wireless gateways. As we have shown in Chapter 5, upgrading such gateways can provide a city-wide coverage of cloudlets. In *vStore*, we consider the heterogeneity of those cloudlet nodes in order to optimize the placement decision. To this end, we implement our framework for Android devices and deploy storage nodes in a city to demonstrate the feasibility and key functionalities of our approach. To the best of our knowledge, this is the first framework that provides the functionality to abstract storage locations and enables storage decisions based on rules that take into account the current context of the user and heterogeneous edge infrastructures.

Summary of contributions. In summary, this chapter makes the following contributions:

- We propose the concept of a middleware that decouples applications from storage infrastructure, motivated by a user study and network measurements (Section 9.3).

- We design and implement *vStore*, a framework that integrates this concept and supports *context-aware micro storage* on heterogeneous cloudlets (Section 9.4). The framework makes storage decisions based on contextual information from the client device and a set of rules that can either be defined globally or customized by the client.

- We conduct a field study in which participants used a demo application for mobile storage. Section 9.5 reports on the insights from this study. The results show that context-based rules are able to reduce the amount of data stored in the cloud. From the obtained insights, we derive future work in Section 9.6.

9.2 Background and Related Work

9.2.1 Context-Awareness

We aim to build a framework that makes storage decisions based on rules that take into account the current context of the user. Context is any information that characterizes a current situation [Dey01] and according to Abowd [Abo+99], a system is context-aware if it uses context to provide relevant information and/or services to its users. In our system, contextual information should influence the storage decision. Examples of relevant context include from where the user retrieves the content, where one is located, or what the network conditions are alike. In general, we can distinguish between low-level context (i.e., raw and unprocessed sensor data) and high-level context that is inferred from (often multiple) low-level context information.

Capturing contextual information on mobile devices. As mobile devices today feature a multitude of built-in sensors, they are able to capture diverse contextual information. The most prominent contextual information is the location. However, it is easy to see how we can extend this to more sophisticated context. Especially fusing data from *hard sensors* (e.g., a GPS receiver or microphone) with data from *soft sensors* (e.g., one's calendar entries) can generate meaningful higher-level context. As an example, assume a user is located at a certain geo-coordinate. Adding a list of point-of-interests, we might derive that he or she is at a sports stadium. Further addition of microphone readings then might derive whether a sporting event is currently in progress. We will later describe how *vStore* uses this kind of context information to make storage decisions.

Usage of contextual information for data placement. Contextual information can be leveraged for making data placement decisions. This is commonly done in CDNs. Besides economic considerations [CZZ13], CDNs typically consider properties of the network (e.g., the topology or link quality) to make placement decisions [Sal+18; Bas+03]. As *user-centric* context, CDNs consider the location distribution of requests [Sce+11] in order to minimize access delays and bandwidth costs. The same holds true for recent approaches that aim at providing storage infrastructure at the edge. As an example, *DataFog* [GXR18] and *FogStore* [GR18; May+17] both perform replication based on spatial locality of data requests.

9.2.2 Mobile Storage

Augmenting (mobile) devices' storage capabilities has mostly been done through Cloud Computing. Research in this direction has for instance proposed specialized file systems that are optimized for the wireless characteristics of device-to-cloud transfers [Don+11]. Following this idea, application-level file systems that consider Edge Computing infrastructure have been developed [Sco+19] but they do not allow fine-grained placement decisions for data based on contextual properties. Tang et al. [Tan+15] suggest uploading data to multiple storage services. However, they only consider cloud storage services. Psaras et al. [Psa+18] suggest buffering data at WiFi access points prior to cloud synchronization.

Complementing cloud storage. Some previous works have proposed to complement cloud storage with an additional layer at the edge of the network. The decision where to store the data is often based on location alone [SMT10], or data is synchronized with cloud storage infrastructures [HL16; Psa+18]. Other than location, network information, and usage patterns of files have been taken into account to make storage decisions [Baz+13; Han+17]. In our work, we do not limit ourselves to these but provide a general concept that operates on rules, which can incorporate whatever contextual information can be gathered by the devices. The *Databox* project [Per+17b; Cha+15; Mor+16] proposes a privacy-preserving intermediate layer between the user's data and the cloud that acts as a mediator to control the usage of the data.

Caching. Several approaches have been proposed for caching data, either in a hierarchical way [Dur+15] or collaboratively determined by content popularity [CP15]. Other works combine caching with prefetching strategies based on predicted mobility [Zha+15a] or for specific applications, e.g., video streaming [Tra+17]. By definition, caching is non-persistent and in our approach, we need higher retention times of the data (e.g., to enable sharing). Hao et al. [Hao+17] present *EdgeCourier*, a system that uses edge devices to synchronize documents. The authors demonstrate the bandwidth saving but such a use case disregards the aspect of sharing data between multiple users. Breitbach et al. [Bre+19] have investigated the joint placement of data and computations in Edge Computing environments. While our approach focuses on data captured by users, they target IoT applications that require large amounts of data as inputs. The authors propose an approach to decouple the placement of such data from the actual tasks, optimizing the tradeoff between execution time and data management overhead.

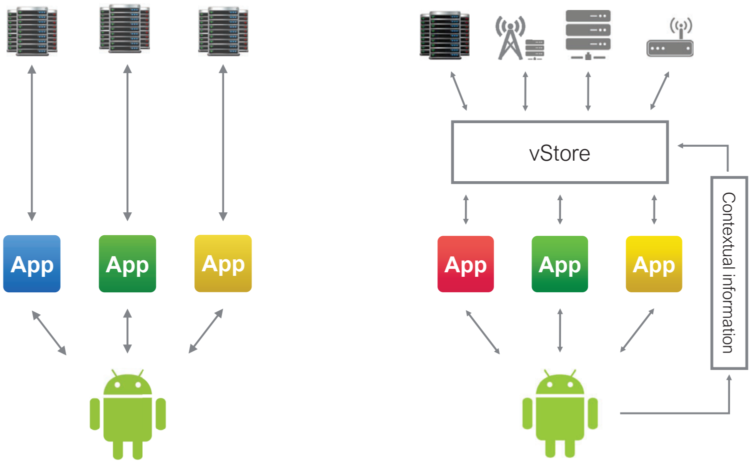
Peer-to-peer approaches. Using peer-to-peer approaches for storage has been proposed in [MRS19; Yan+10; CLP17]. Pure ad-hoc approaches as presented in [Pan+13b] are not feasible in practice, as our client devices are assumed to be highly mobile and, thus, cannot guarantee spatial locality of the data. Furthermore, if we assume cellular connections, they typically have a low uplink bandwidth, making the retrieval of data slow for other peers. Yang et al. [Yan+10] assume that the data originates from the cloud and is replicated at cloudlets at the edge. This is opposite to the data flow of a reverse CDN as we assume it to be the case in our use cases. Confais, Lebre and Parrein [CLP17] extend the design of the IPFS protocol [Ben14] to support edge and fog nodes. Monga et al. [MRS19] present a federated store for streams of data blocks that emphasizes on reliability. None of these approaches however enable the same flexible context-aware placement as our rule-based approach.

Replication. Some works have investigated replication strategies that are built on top of heavyweight distributed data storage systems (*Apache Cassandra*) [GR18; May+17; GXR18], making the practical deployment on constrained edge nodes questionable. When replicating data, issues w.r.t. consistency can arise [Mor+18c; Mor+18d]. This is beyond the scope of our concept system, and we assume all stored data to be immutable.

Summary. While offloading computations closer to the edge of the network has been studied extensively, the possibility to extend data storage towards the edge has seldom been examined. Existing works mostly use cloudlets at the edge as a buffer for cloud synchronization or as a cache for data originating either from the cloud or from IoT devices. Most importantly, except for a few approaches that consider the location of users and data, the storage decision is agnostic to contextual properties, and does not support sharing of data. In contrast, we propose a concept for a sharing-enabled, context-aware edge storage for data that is captured with personal mobile devices.

9.3 Context-Aware Storage at the Edge

We propose a novel approach to provide context-aware micro-storage to mobile users as opposed to using inflexible cloud storage locations that do not take into account the user's context for placement decisions. Figure 9.1 contrasts these two approaches, with Figure 9.1(a) showing the traditional approach, where all data is stored in homogeneous cloud environments. The storage location is determined by the individual application. In contrast to that, we propose *vStore* as a middleware to abstract from predefined storage locations (Figure 9.1(b)). Requests to store and retrieve data from the client are handled by the middleware. Hence, vStore enables decoupling between the individual applications and the location where the application's data is stored. Instead of solely relying on cloud infrastructures, the middleware supports different types of storage nodes.



(a) Traditional application-specific cloud storage

(b) Our proposed concept

FIGURE 9.1: COMPARISON OF STORAGE APPROACHES†

9.3.1 Motivation and Use Cases

To further justify the need for vStore, we describe three use cases that benefit from our approach. These use cases are grounded in a survey we conducted. In total, the survey had 51 participants. The participants were aged 16–40 and mostly students and researchers. This survey helped to (i) understand current usage patterns and challenges encountered by mobile users w.r.t. data storage and retrieval and (ii) derive requirements for our storage framework. In the following, we highlight essential insights about how people capture, use, and share data generated with their phones. All survey questions can be found in Appendix H. Throughout this section, questions are referenced by their number as listed in the appendix (e.g., Q1).

9.3.1.a Sharing data at an event

Especially during large-scale events, cellular networks are often congested [Frö+16]. A prominent example are football matches. Figure 9.2 shows measurements of the available cellular bandwidth during a match at the Commerzbank Arena, a stadium in Frankfurt (Germany) with a capacity of 51 500 spectators. We measured the available bandwidth to a cloud location using the *iperf*[2] tool. Compared to the average bandwidth available in the stadium when no match takes place, we can clearly see that the network quality decreases tremendously. At some distinct events, such as goals occurring in the match, the network collapses almost entirely. At half-time, the network became entirely unavailable. In such cases, edge cloudlets (that for instance are deployed on several WiFi access points) can be useful to provide users with storage services. Besides the obvious use case of storing data in the cloud for later use or sharing with people not present at the event, a more interesting use case for edge storage arises when data is to be shared among people present at the very same event. This type of sharing has been examined before in the context of video streaming [Dez+12] but not with the support of Edge Computing infrastructure.

In our survey, over 50 percent of the participants stated that they at least occasionally share data such as pictures at an event (Q17). About 20 percent of the time, sharing is done with other people attending the same event (Q20). Only 4 percent of our participants have never experienced congested connections during events (Q18).

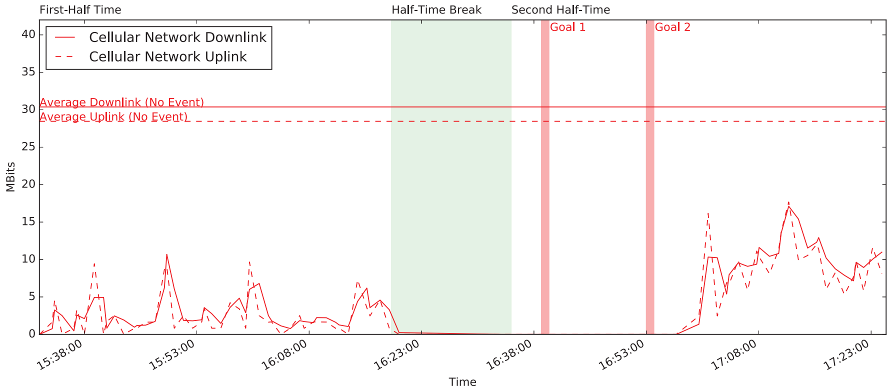
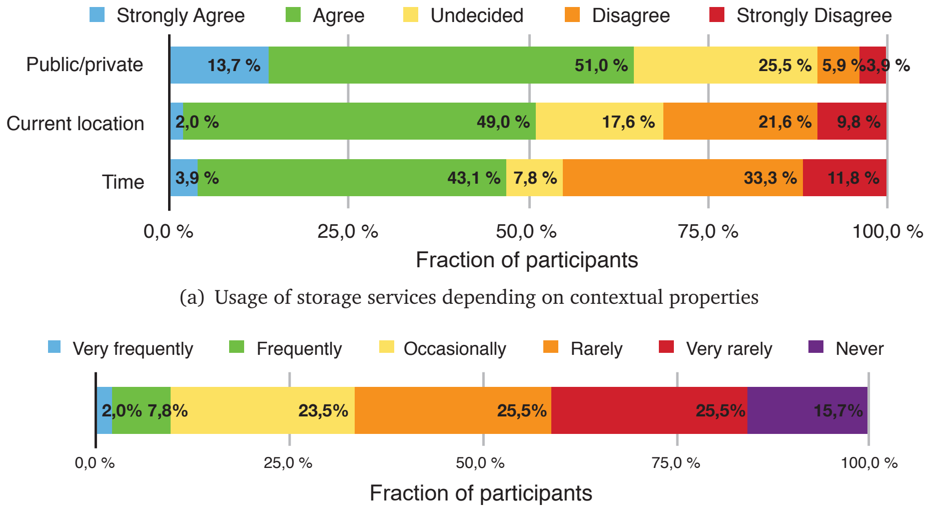


Figure 9.2: Measured cellular bandwidth during a football match†

[2] https://iperf.fr/ (accessed: 2020-03-09)

9.3.1.b Context-aware storage across applications

In our survey, we questioned participants whether the storage services they choose to use depend on (i) whether the data is intended for private or public use, (ii) their current location, and (iii) the date and time of data capture. The results of those questions are depicted in Figure 9.3(a). We can clearly observe that the majority of users base the decision on where to store their data to a great extent on these three contextual properties. Following this result, these contextual properties, among others, will be used by our framework to make storage decisions. Furthermore, some users upload the same data to more than one storage service (see Figure 9.3(b)). With regards to this result, vStore offers the opportunity to provide a unified interface through which users access different storage services.



(a) Usage of storage services depending on contextual properties

(b) Upload of the same data to multiple storage services

FIGURE 9.3: SURVEY RESULTS ABOUT THE CONTEXT-DEPENDENCE OF CHOOSING STORAGE SERVICES AND USING MULTIPLE STORAGE SERVICES†

9.3.1.c Getting suggestions for data related to one's current context

When at a certain location or when performing a certain activity, users often search for information related to that specific context. With the capability to query our framework for data that is similar to one's usage context, we can provide users with this kind of information. Coming back to the example of an event, over 78 percent of our surveyed participants at least sometimes retrieve data related to an event they attend (Q19). Furthermore, retrieving context-aware information is a crucial building block for augmented reality applications, as shown in [Mül+17].

9.3.2 Problem Definition and Requirements

From the use cases described above, we define the problem we want to tackle as follows: given data that is captured by mobile users and contextual information, determine a location where the data should be stored. Besides the device itself and cloud storage, various storage locations at the edge should be considered. Storage locations should best reflect the future retrieval patterns (e.g., where and when) of the data. Furthermore, the decision should also consider whether the data is intended for private use or sharing.

In order to provide context-aware micro-storage as we envisioned, the system should fulfill the following requirements: (i) storage location agnosticism, (ii) openness to extensions (e.g., allowing to incorporate more contextual properties) and third-party applications, and (iii) extensibility to implement new rules for storage decisions. In the next section, we will describe the design of a system that realizes our concept and outline how it fulfills these requirements.

9.4 System Design and Implementation

In this section, we describe the design of our system and its individual components. Figure 9.4 shows a high-level overview of our system. Our proposed concept is composed of several building blocks: (i) the *vStore* framework that provides interfaces to applications and storage nodes, and makes storage decisions, (ii) individual *storage nodes* on which data can be stored, and (iii) a *master node* that maintains a global view on the storage locations of data items. In addition, *context providers* provide contextual information for the storage decisions and an external *configuration file* defines basic settings for the operation of the framework. In the following subsections, we will explain those building blocks in more detail, and present a demo application that makes use of *vStore* on Android phones.

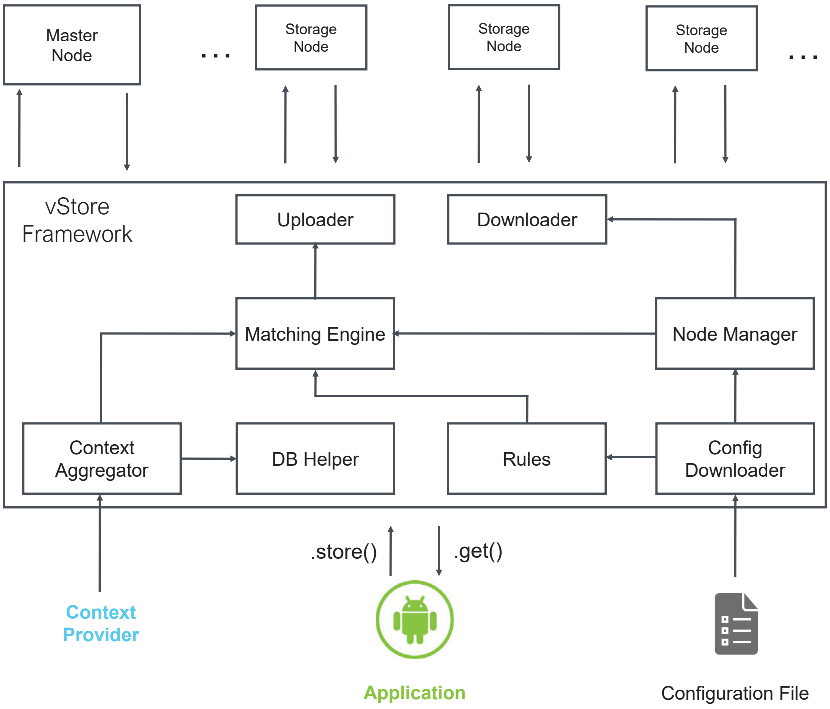


FIGURE 9.4: SYSTEM ARCHITECTURE†

9.4.1 vStore Framework

The central contribution of this chapter is the concept of *vStore*, a framework that provides interfaces to applications in order to store and retrieve data while abstracting from a concrete storage location. The framework collects current contextual information, maintains a list of available storage nodes and—based on a set of rules—makes the decision where to store the data. For each data item to be stored, a unique identifier is generated that is later used to retrieve specific data across storage nodes. The framework is implemented as a Java Library for the Android operating system. Figure I.1 in Appendix I shows the complete class diagram of the implementation.

9.4.1.a Context aggregator and context provider

The task of the context aggregator is to collect the different kinds of contextual information. It supports different types of contexts that can be supplied by *context providers*. As a general way to supply contextual information, context providers can use a key-value representation of the data in a JSON file. This is a common way to represent contextual information in the IoT domain [Per+14a]. Context providers can *push* contextual information to the context aggregator via a message bus. Alternatively, through an API that must be provided by the context provider, the context aggregator can *pull* the currently available contextual information from the provider. Besides this general interface, the context aggregator has built-in support for various contextual information provided by the APIs of the Android operating system. In Figure 9.5, the architecture of the aggregator is summarized.

To gather contextual information from the mobile phones, we rely on three providers of such information: First, we make use of *AWARE*[3], an open source framework for context instrumentation on Android phones. Second, the Google Places API provides a list of places that surround the user, their type, and the likelihood of the users being located at those places. Third, the Android Connectivity API provides information about the network connectivity of the device. In the following, we list the different types of context supported by the context aggregator and how they are acquired in our implementation:

Location | A plugin for AWARE provides location information using the *Google Fused Location API*[4].

Places | Whenever a new location is available, we query Google's *Places API*[5] for an updated list of places. We group the large number of place types provided by this API into three groups, namely points of interest (POI), events (such as stadiums, city halls, and night clubs) and social places (such as restaurants, cafes, and bars).

Noise | The ambient noise level is measured by an AWARE plugin through the phone's microphone. By configuring a threshold, we can determine if the current environment should be considered as loud or silent.

[3] https://www.awareframework.com (accessed: 2020-03-27)

[4] https://developers.google.com/location-context/fused-location-provider (accessed: 2020-03-21)

[5] https://developers.google.com/places/web-service/intro (accessed: 2020-03-21)

ACTIVITY | The user activity is provided by another plugin that internally uses the Google Awareness API to identify the user's current activity (e.g., idle, driving, or walking).

NETWORK | We use Android's *ConnectivityManager* and *TelephonyManager* to fetch details about the user's current connectivity state (e.g. to what kind of network the user is currently connected to).

DATE AND TIME | The time and date as reported by the phone's operating system.

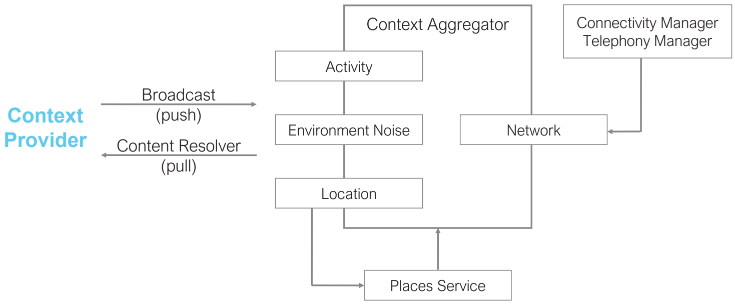


FIGURE 9.5: CONTEXT AGGREGATOR[†]

9.4.1.b Node manager

As outlined at the beginning of this section, the core of our concept is to make a sensible placement decision for data, given different available storage nodes. The node manager maintains a list of all available storage nodes. When storage nodes are added to the framework, their type, location and bandwidth need to be specified. Available nodes can then be queried according to these properties. Before a node is stored in the internal database of vStore, the node manager checks if the node is reachable. Node information can be updated and deleted through an API.

9.4.1.c Rules

In our framework, rules are used to make the storage decision and are evaluated by the matching engine, as described in Section 9.4.1.d. Rules can either be defined globally by a configuration file that is remotely pulled (see Section 9.4.4) or created individually by users. In detail, our rules consist of three parts that will be evaluated in the following order during the matching:

(i) METADATA PROPERTIES | These denote for which MIME type and file size the rule should be applied during the matching process.

(ii) CONTEXT FILTERS | Context filters determine which contextual properties must be fulfilled for the rule to be applied. Any of the aforementioned contextual information can be specified here. Depending on the type of context, different filter operations can be used. For example, using the places context, we can define a binary filter that checks if a user is at a certain location. A contrasting example is the date and time, or noise levels, where we can specify a certain range or threshold value.

(iii) STORAGE DECISION LAYERS | The decision layers determine which storage nodes are chosen. Rules can include many of those layers, ordered by priority. Each layer defines constraints on the storage node, such as available bandwidth or distance to the user. A decision layer can also point to one specific storage node. In this case, the file will be stored on that specified node. Optionally, and for future work on replication, we also include the definition of a minimum number of replicas.

The way we define rules follows the *Event Condition Action* (ECA) paradigm. We chose to follow the ECA principles for the design of our rules because ECA rules are a common structure in the domain of *active databases* [PD99; HS02]. Mapped to our implementation, an event represents the storage request of a file. Conditions are the contextual filters of the rule that have to be fulfilled, and the action is the selection of storage nodes by the decision layers.

9.4.1.d Matching engine

The matching engine is the main part of the framework. Given a file f, the current context C, and a set of available storage nodes N and rules R, it decides on which storage node the file will be saved. Hence, this storage decision function can be formalized as:

$$(f \times C \times N \times R) \rightarrow N$$

The matching process consists of four main steps. The pseudocode is shown in Algorithm 2. As a first step, only rules that match file metadata (type and size) are considered (line 1). For instance, a rule that only applied to image files would not be evaluated further if the data the user wants to store is a document. The same applies if a rule specifies that it should only be applied to files of a certain size. Next, all contextual filters as defined by the rule have to match the context that is given at the time of evaluating the rule. Only those rules are triggered and evaluated further (lines 2–6).

In the third step of the matching process, for each of the remaining rules that satisfy the metadata and context filters, a *detail score* $s \in \mathbb{R}, 0 \leq s \leq 1$ is computed to determine the rule that most accurately describes the current context. Two factors influence this score: (i) how many contextual filters are defined in the rule, and (ii) for continuous filters (i.e., that operate on a certain range) how narrow the filter is. For instance, a rule that triggers within 150 meters of a point of interest would be assigned a higher score than one triggering within 500 meters. For each contextual property, we define a maximum value to which the property can contribute to the overall detail score. A higher value means that this contextual property is considered to be more important. Table 9.1 shows the maximum detail score per contextual property and whether it is discrete (i.e., a binary filter that describes if a contextual property is present or not) or continuous (i.e., a range of possible values). In the latter case, a function maps the size of the range to the maximum allowed detail score. The total detail score is then given by the sum of all scores per contextual property. According to this metric, the most detailed rule is chosen to be executed (line 7). We chose these maximum detail scores such that they emphasize use cases that require location context (including places) and the date and time. For example, this allows to accurately represent use cases as described in Section 9.3.1.a.

Lastly, when a rule is selected, we iterate over the decision layers (lines 8–14) and available storage nodes (lines 9–13). For each pair of decision layers and storage nodes, traversed in order of the decision layers, we check if the rule's constraints w.r.t. bandwidth and distance match the node's properties (line 10). If that is the case, this node is chosen and returned as the storage location (line 11).

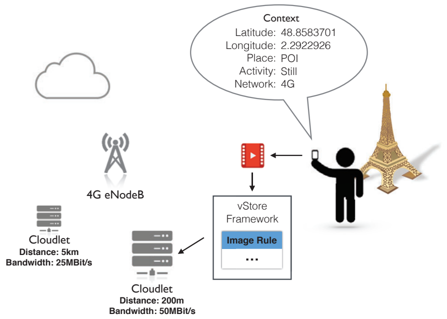
Algorithm 2 Storage matching

```
    Input file, context, N, R
    Output $n \in N$
 1: rules ← getRulesMatchingMetadata (R, f.size, f.type)
 2: for $r \in rules$ do
 3:   if $\wedge_{c \in r.getContextFilters}$ matchesContext?(c,context) then
 4:     triggeredRules.add(r)
 5:   end if
 6: end for
 7: selectedRule ← getRuleWithHighestDetailScore(triggeredRules)
 8: for (dl ∈ selectedRule.getNextInDecisionLayer() do
 9:   for n ∈ $N$ do
10:     if dl.bandwidth.matches?(n.bandwidth) ∧
        dl.distance.matches?(n.distance) then
11:       return n
12:     end if
13:   end for
14: end for
```

Table 9.1: Detail scores per contextual property

Contextual property	Max. detail score	Mapping
Location	0.2	continuous
Places	0.15	continuous
Date and time	0.15	continuous
Sharing domain	0.1	discrete
Activity	0.1	discrete
Network	0.1	discrete
Noise	0.1	discrete

Figure 9.6 shows an example of a storage decision, in which a user takes a picture at a point of interest. Rules that match the current context are triggered, i.e., in this example, the user is idle and not in motion, connected to a 4G network, and at a point of interest. In this example, the rule labeled *Image Rule* in the figure is the one with the highest score, and in its decision layer, a cloudlet with a maximum distance of 200 meters and at least 50 MBit/s of bandwidth is chosen to store the picture.



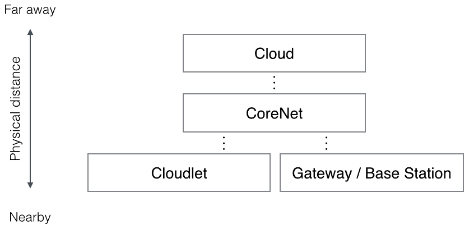


FIGURE 9.6: EXAMPLE OF RULE MATCHING† FIGURE 9.7: STORAGE NODE HIERARCHY†

9.4.2 Storage Nodes

Storage nodes are the devices that are available to store the data. In a real-world deployment, a storage node could be hosted on a variety of devices, either close-by or distant to the user. To take into account this heterogeneity, vStore defines different types of storage nodes as depicted in Figure 9.7. Besides cloud nodes, we consider cloudlets, gateway nodes, and nodes in the core network. Gateway nodes are devices to which users have a direct wireless connection, such as WiFi access points or cellular base stations. In addition, we also consider private clouds as a type of storage nodes, i.e., deployments that are owned by end users themselves. One example of such a system is *ownCloud*[6]. Including this kind of nodes is sensible for the definition of storage rules aimed at the storage of private data, i.e., data that is not shared among different users of the framework.

Storage nodes are registered to the global configuration file (see Section 9.4.4) and master node (see Section 9.4.3) with their available bandwidth and location. For our prototype implementation, we use the geographic locations of the nodes, assuming that physical proximity correlates with network latency. We note that for future work, network coordinates (such as Vivaldi coordinates [Dab+04]) could give a more realistic estimation of the quality of the connection to the storage nodes.

9.4.3 Master Node

When a file is saved through the framework, a global identifier (using an UUID) is generated. This UUID acts as an identifier when requesting a previously stored file. The clients that originally uploaded the file to the framework keep a record of the nodes their file was stored on. In addition, for each storage decision, the master node keeps a record that contains the mapping of the UUID to the storage node(s) on which the file is stored. This is required for two reasons: (i) in the case of sharing the file, other clients do not know the storage location a priori, and (ii) in the case of replication and failure of one storage node, clients need a list of all storage nodes that have copies of the file. Recall that the implementation of our concept does not consider mutable data and therefore, common challenges found in distributed file systems, such as providing consistency guarantees, are beyond the scope of our contribution. Existing works like the Google File System [GGL03] have developed mechanisms where a master node interacts with several storage nodes

[6] https://owncloud.org (accessed: 2020-03-27)

and provides fault tolerance and consistency guarantees. We leave the integration of such concepts into our approach for future work.

9.4.4 Configuration

The framework can be configured externally. This mainly serves two purposes: (i) initially retrieving available storage nodes, and (ii) including global rules for the placement decision. Defining global rules that are available on all devices is important for users who do not wish to specify custom rules. This ensures that at least some basic storage decisions can be made. Furthermore, this could allow service providers to update rules, e.g., in order to react to changes in network conditions or node availability.

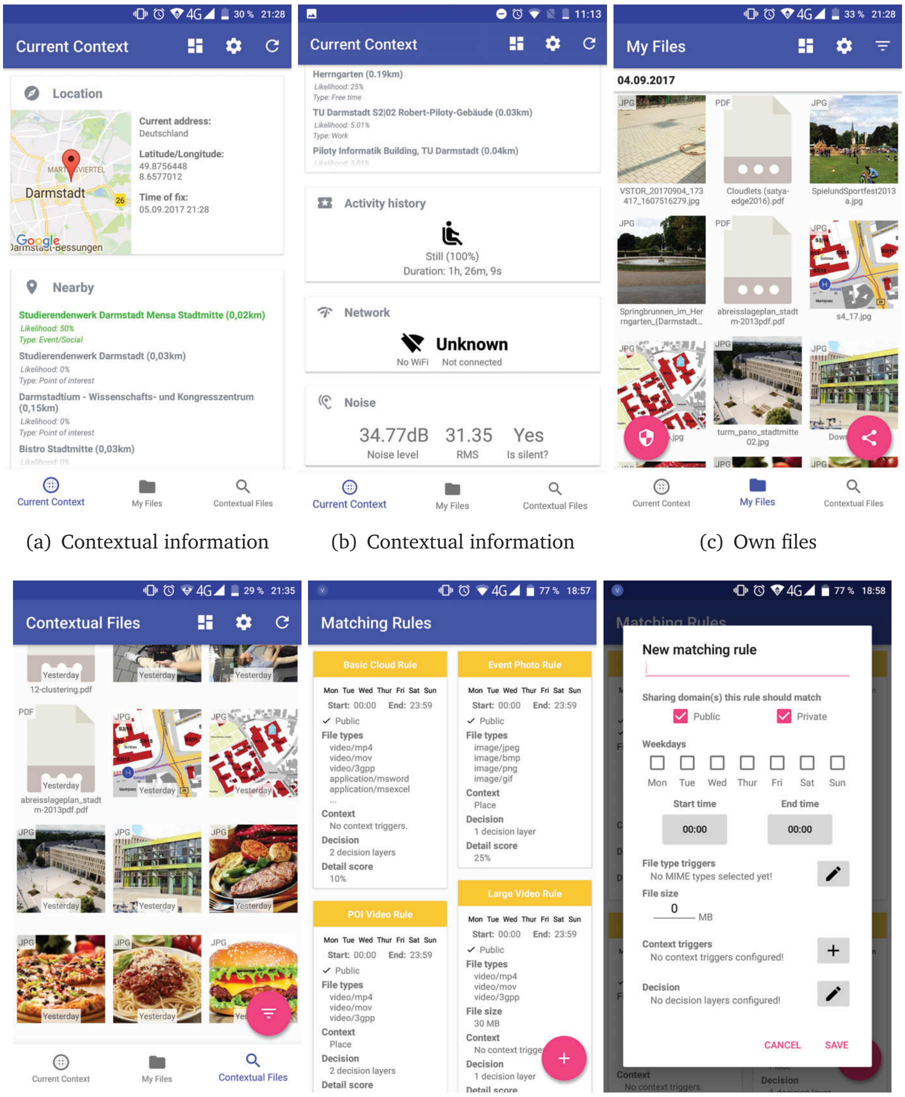
To this end, for simplicity reasons, the framework uses a central configuration file that can be pulled from a cloud location. This provides a mechanism for retrieving and updating available storage nodes and rules. In the future, we envision the configuration of the framework to be managed in a distributed way and to be pushed to the devices whenever new information is available. This would for instance enable users who have the same or similar context to share custom rules they have defined.

9.4.5 Demo Application

To conduct our field study (see Section 9.5) we developed an application for the Android platform that uses our framework. This application provides users the possibility to store, view, and retrieve files; similar to applications for cloud storage. The functionality allows to represent use cases as described in Section 9.3.1. In detail, the application allows users to (i) store their data on a storage node determined by the matching engine of the framework (enabling context-awareness as required for use cases like the one described in Section 9.3.1.a), (ii) define whether this data should be accessible publicly or not (allowing sharing of data, see Section 9.3.1.b), (iii) view and create custom storage rules, and (iv) retrieve context-related data from other users (see Section 9.3.1.c). Information about locally stored files, available storage rules, and nodes is stored in an *SQLite* database[7]. The database scheme is shown in Figure I.2 of Appendix I.

Figure 9.8 depicts the main screens of the application. The application's main screen shows a summary of all current contextual information available (Figures 9.8(a) and 9.8(b)). The user's files are shown in a grid-layout as seen in Figure 9.8(c). For files that are photos or videos, a thumbnail is shown to preview the contents of the file. Tapping the file opens it in the default application for that file type. On the same screen, users can also add new files, by tapping one of the two pink buttons in the lower part of the screen. The left button (depicting a shield) is used to add private files (i.e., files that cannot be retrieved by other users) while the right one is used to add files that are to be shared. Besides their own files, users can also retrieve files based on contextual queries (e.g., files that were captured nearby or at similar places). Figure 9.8(d) shows an example of contextually similar files. The pink icon in the lower right corner allows users to customize the context filters, i.e., users can choose which contextual properties should be queried. Note that in order to save bandwidth, initially, only thumbnails are downloaded. The entire file

[7] https://www.sqlite.org/index.html (accessed: 2020-03-27)



Current Context

Location

Current address:
Deutschland

Latitude/Longitude:
49.8756448
8.6577012

Time of fix:
05.09.2017 21:28

Nearby

Studierendenwerk Darmstadt Mensa Stadtmitte (0,02km)
Likelihood: 50%
Type: Event/Social

Studierendenwerk Darmstadt (0,03km)
Likelihood: 0%
Type: Point of interest

Darmstadtium - Wissenschafts- und Kongresszentrum (0,15km)
Likelihood: 0%
Type: Point of interest

Bistro Stadtmitte (0,03km)
Likelihood: 0%

(a) Contextual information

Current Context

Herrngarten (0.19km)
Likelihood: 25%
Type: Free time

TU Darmstadt S2|02 Robert-Piloty-Gebäude (0.03km)
Likelihood: 5.01%
Type: Work

Piloty Informatik Building, TU Darmstadt (0.04km)
Likelihood: 5.01%

Activity history

Still (100%)
Duration: 1h, 26m, 9s

Network

Unknown
No WiFi Not connected

Noise

34.77dB Noise level
31.35 RMS
Yes Is silent?

(b) Contextual information

My Files

04.09.2017

VSTOR_20170904_173417_1607516279.jpg
Cloudlets (satya-edge2016).pdf
SpielundSportfest2013a.jpg
Springbrunnen_im_Herrngarten_(Darmstadt...
abreisslageplan_stadtm-2013pdf.pdf
s4_17.jpg
turm_pano_stadtmitte02.jpg

(c) Own files

Contextual Files

12-clustering.pdf
abreisslageplan_stadtm-2013pdf.pdf

(d) Contextual files

Matching Rules

Basic Cloud Rule
Mon Tue Wed Thur Fri Sat Sun
Start: 00:00 End: 23:59
✓ Public
File types
video/mp4
video/mov
video/3gpp
application/msword
application/msexcel
...
Context
No context triggers.
Decision
2 decision layers
Detail score
10%

Event Photo Rule
Mon Tue Wed Thur Fri Sat Sun
Start: 00:00 End: 23:59
✓ Public
File types
image/jpeg
image/bmp
image/png
image/gif
Context
Place
Decision
1 decision layer
Detail score
25%

POI Video Rule
Mon Tue Wed Thur Fri Sat Sun
Start: 00:00 End: 23:59
✓ Public
File types
video/mp4
video/mov
video/3gpp
Context
Place
Decision
2 decision layers
Detail score

Large Video Rule
Mon Tue Wed Thur Fri Sat Sun
Start: 00:00 End: 23:59
✓ Public
File types
video/mp4
video/mov
video/3gpp
File size
30 MB
Context
No context trigge
Decision
1 decision layer
Detail score

(e) List of rules

New matching rule

Sharing domain(s) this rule should match
☑ Public ☑ Private

Weekdays
□ Mon □ Tue □ Wed □ Thur □ Fri □ Sat □ Sun

Start time 00:00
End time 00:00

File type triggers
No MIME types selected yet!

File size
0 MB

Context triggers
No context triggers configured!

Decision
No decision layers configured!

CANCEL SAVE

(f) Custom rule creation

FIGURE 9.8: SCREENSHOTS OF THE DEMO APPLICATION†

is retrieved only when the users select the file to be viewed. The application furthermore provides a user-facing interface to the framework storage rules. Figure 9.8(e) shows all currently active storage rules. Tapping on the pink icon in the lower right corner opens up the interface shown in Figure 9.8(f) that allows the creation of custom rules.

After having carried out a field trial using this app, users were asked to assess the usability of the demo application using the *System Usability Scale* (SUS) [Bro96]. According to the results, the application has an average SUS of 76.6[8].

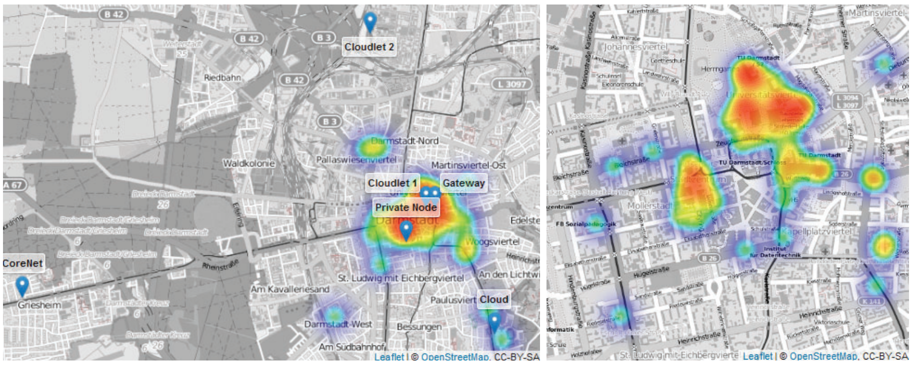
[8] The SUS ranges from 0 (worst) to 100 (best). According to [BKM09], our SUS of 76.6 ranks between "good" and "excellent".

9.5 Experience Report

In this section, we report on experiments we conducted using the demo application we described in Section 9.4.5. We show the feasibility of our approach by deploying several storage nodes and conduct a field study by defining sample rules and evaluating the resulting storage decisions that *vStore* made. The results of the field study allow us to gain further insights on which contextual properties, and therefore which kinds of rules, are relevant in practice.

9.5.1 Experimental Setup

Storage nodes. We deployed a total of six storage nodes in the area of Darmstadt, Germany. The maps shown in Figure 9.9 visualize our deployment. Figure 9.9(a) shows an overview of the area with the location of the storage nodes and Figure 9.9(b) zooms in on the city center with a heatmap depicting where most of the data was captured. For a rapid deployment, we used a RaspberryPi (version 3, model B) to host the storage nodes and MongoDB as a database to store the data. A NodeJS server implements the storage service and acts as an interface between MongoDB and the *vStore* framework. To simulate different types of storage nodes we would have in a large-scale deployment, we set different node types in our system: two cloudlets, one gateway, one cloud node, one core net node, and one private cloud.



(a) Overview

(b) City center

Figure 9.9: Node locations and usage heatmaps†

Rules. We defined several global rules that were pushed to the phones in our field trial of *vStore*. The most relevant are listed in Table 9.2. Note that we omit the detailed description of other rules that were active and mostly used to test certain features (e.g., one rule triggered at an exact location placed the storage on the phone only). The table shows the contextual properties that have to match according to the rule, as well as the detail score of the rule. The bottom row furthermore reports how often the rule was executed in our field trial.

The *POI Photo Rule* is executed when a user is near a point of interest and wants to upload a photo. It is applied to files of any size and only for data that is to be shared, reaching a detail score of 25 %. The decision layer first attempts to save the file on a gateway node within 5 km of the user's location, otherwise, a cloudlet within 20 km is used. The *Social Photo Rule* is executed at locations that are tagged *social* according to the places API. This rule has the same contextual filters as the previous POI photo rule and the same detail score. We define these two rules to be able to evaluate them separately, according to the different place contexts. The *Driving Rule* is applied when the context aggregator reports the user's activity as driving. This rule would also be triggered if a user is aboard a train or bus. Any file uploaded in this context will not be stored on nearby cloudlets since the user might only drive by a nearby POI without the intention of sharing or retrieving related data. Therefore, we define this rule such that files will be stored in the cloud. The *Event Photo Rule* combines two different contextual filters. First, a user has to be at a place that is of the type event. Second, a certain noise level has to be captured. We determined this value empirically by comparing readings of the AWARE noise plugin with perceived ambient noise levels. The resulting threshold of 20 dB seems to point towards calibration errors of the plugin, but this value seemed to represent loud environments. The rule stores photos that are to be shared on a cloudlet within a radius of 30 km. The *Basic Cloud Rule* is used as a fallback due to the low detail score, should no other rule yield a result. It then checks if a core net node with a bandwidth of 10 GBit/s is available to upload the data. If this is not the case, the file will be stored in the cloud. To evaluate the storage of private files, we create the *Basic Private Rule*. This rule stores all files that are not intended for sharing on a private storage node.

Users. We distributed the demo application to six participants and configured the framework with the aforementioned rules. The participants were asked to use the application to capture various kinds of data (e.g., photos, videos, contacts) and store them using the demo application described in Section 9.4.5.

9.5.2 Usage Patterns and Storage Decisions

We now look at how users used the application, i.e., which types of data they stored and which storage decisions were made based on the rules we defined. The bottom row of Table 9.2 shows how many times each rule was triggered. All our defined rules were triggered during the user study. We can observe that the *Basic Cloud Rule* was triggered the most, however, data was stored on cloud nodes only for 29.3 % of all data. This is because the cloud rule has a very low detail score. In many cases, other rules that relate to the user's location or define proximity to a point of interest, have a more detailed score and therefore those are the ones that determine the placement. We can think of the cloud rule as a fallback, in case there is no most likely place (e.g., when we are not sure where the user is).

The resulting placement decisions for the different file types that users captured during our study are shown in Table 9.4. The results are listed per individual storage node. In total, users stored 178 files using vStore, most of which were photos. Out of those, 35.9 % were stored on cloudlets, 19.3 % on gateway nodes, and 2.7 % on core net nodes. In contrast to this, without vStore, users would likely have all their photos uploaded to distant cloud infrastructures. These numbers confirm the benefits that can be obtained in future Edge Computing environments.

Table 9.3 lists the sharing ratio for each type of data, i.e., whether users marked the data to be publicly shared on the storage nodes or for their private use. From the results, we can observe that the sharing ratio heavily depends on the data type.

TABLE 9.2: PLACEMENT RULES[†]

	POI Photo	**Social Photo**	**Driving Rule**	**Event Photo**	**Basic Cloud**	**Basic Private**
Context	*Place:* POI	*Place:* Social	*Activity:* Driving	*Place:* Event, *Noise:* $\geq$-20 dB	None	None
File size	Any	Any	Any	Any	Any	Any
File types	JPG, BMP, PNG, GIF	JPG, BMP, PNG, GIF	Any	JPG, BMP, PNG, GIF	Any	Any
Sharing domain	Public	Public	Public	Public	Public	Private
Days	Mon–Sun	Mon–Sun	Mon–Sun	Mon–Sun	Mon–Sun	Mon–Sun
Time	Any	Any	Any	Any	Any	Any
Decision layers	**Layer 1** Gateway $\leq$ 5 km **Layer 2** Cloudlet $\leq$ 10 km	**Layer 1** Cloudlet $\leq$ 5 km **Layer 2** Cloudlet $\leq$ 10 km	**Layer 1** Cloud	**Layer 1** Cloudlet $\leq$ 30 km	**Layer 1** CoreNet ↑ 10 GBits/s ↓ 10 GBits/s **Layer 2** Cloud	**Layer 1** Private-node
Detail score	0.25	0.25	0.2	0.35	0.1	0.1
Times triggered	36	34	18	5	47	5

While users were willing to share over 80 percent of their images, for more sensitive information such as contacts this number drops down close to 3 percent. With the set of rules we defined, we are able to capture the user's intention in this aspect, as the sharing domain influences the placement decision vStore makes.

TABLE 9.3: TOTAL NUMBER AND SHARING RATIO OF DATA TYPES

	Total number of files	**Sharing ratio**
Image	145	81.46 %
Video	17	9.55 %
Document	11	6.18 %
Contact	5	2.81 %

9.5.3 Discussion

With our preliminary experiments outlined in this section, we were able to show how we can couple the placement of data to the context of the user. To do so, we used a rule-based matching that includes heterogeneous storage nodes at the edge.

TABLE 9.4: PLACEMENT RESULTS BY LOCATION AND DATA TYPE

		Type of Data				
		Image	Video	Document	Contact	Σ
Storage location	Gateway	28	3	0	2	33
	Cloudlet 1	35	6	1	0	42
	Cloudlet 2	17	1	0	0	18
	CoreNet	4	1	2	0	7
	Cloud	43	3	7	3	56
	Private Node	4	0	1	0	5
	Phone	14	3	0	0	17

The results showed that our rules were able to capture usage contexts that led to placement decisions closer to the edge, e.g, on cloudlets or gateway nodes. Figure 9.10 summarizes the number of times a file was saved on each type of storage node in our field trial. From the figure, we can observe that—in line with the relationship between Cloud Computing and Edge Computing described in Chapter 2—in practice, Edge Computing will complement Cloud Computing, as suggested by the near equal number of cloud locations that were chosen compared to cloudlets. In Table 9.4, we can observe that our storage rules were able to represent natural choices for data storage locations. As an example, for videos, cloudlets were the most used storage destination, while for documents (data that has lower bandwidth and latency requirements, and likely lower spatio-temporal access correlations) the cloud was storage location most often used. Having appropriate storage rules and accurate contextual information therefore allows to make these decisions automatically.

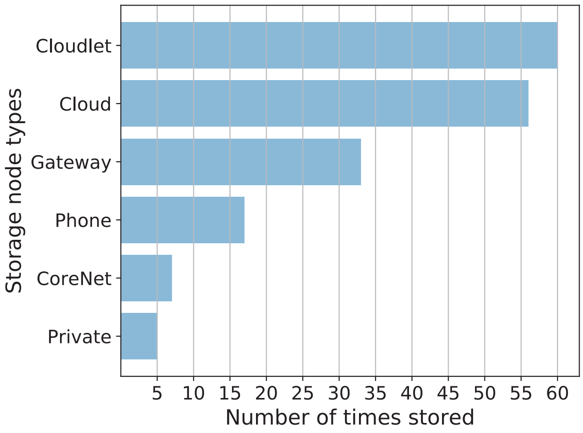


FIGURE 9.10: NUMBER OF PLACEMENTS PER STORAGE NODE TYPE

However, the accuracy of contextual descriptions remains an issue. For instance, files were sometimes saved using an incorrect context, due to the fact that the context is not updated in real-time. Keeping an accurate context on a mobile phone remains a trade-off between accuracy and energy consumption. In addition, much work still needs to be done in order to correctly recognize higher-level context.

Nevertheless, our user study exemplified the usage of cloudlets, especially if they are located at the edge of the network and close-by to mobile users. This is especially true in the context of sharing data locally. For this use case, *vStore* offers the possibility to define rules that are triggered when a user is at a certain location or point of interest. As outlined in Section 9.3.1.a, many people today share data at events, some of them even with people present at the same event. Our results further demonstrated a general high sharing ratio, irrespective of ongoing events (see Table 9.3). For the future, we envision storage cloudlets to be deployed throughout city areas, some of which will be co-located at the radio access network or act as gateway nodes themselves (e.g., WiFi hotspots during events).

Of course, appropriate rules are required to make the framework beneficial in practical use. We enable users to define custom rules for representing their usage intentions. In addition to custom rules, the framework allows for global rules to be configured. In our field trial, we could see that even with just a basic set of global rules, these were often executed when making the placement decisions. In future use of the system, infrastructure providers could set these global rules, e.g., to specify local cloudlets on gateway nodes when regular networks are overloaded.

9.6 Conclusion and Future Work

In this chapter, we have extended the placement problem in Edge Computing from functional application parts to data that users capture with their mobile devices. We have motivated the need for *context-aware micro-storage* for mobile users with a user study and by measurements of available cellular network bandwidth during a large-scale event. The user study gave valuable insights about usage patterns of mobile users with regards to data storage. We then presented the concept of *vStore* (virtual store), a framework that implements the building blocks for enabling micro-storage at the edge of the network. Our concept (i) enables the decoupling of storage location from predefined cloud locations, (ii) leverages small-scale cloudlets at the edge of the network to provide proximate storage locations, (iii) uses various contextual information to make the placement decision, and (iv) allows for cross-application sharing of data. A set of global and local rules allows different stakeholders (e.g., end users or infrastructure providers) to define custom rules that are evaluated when making the decision where to store the data. The concept of context-aware storage decisions allow to better support various use cases, e.g., sharing data at events through proximate storage cloudlets (see Section 9.3.1.a). As the number of consumer devices that generate data increases, storing and retrieving data from nearby cloudlets saves scarce core network bandwidth. The concept presented in this chapter therefore is an important building block for Edge Computing in the context of data-centric applications.

We conducted a field study of our system using an application for the Android platform through which users could capture and upload data. Furthermore, users were able to retrieve data related to their current usage context. We deployed different storage nodes in a major city and through the implementation of example decision rules we were able to show how this framework can complement existing cloud-based storage infrastructures. Specifically, we showed we could reduce the number of times that files were saved in the cloud. This could be achieved based on rules that contain context mappings (e.g., one's location and sharing intention).

The framework presented here opens up a lot of opportunities for future work. We make all components of the framework[9,10,11] and the demo application[12] publicly available as open source to encourage future research. In particular, we suggest investigating the following in future work:

Replication | Our framework assumes a single location where the data will be stored. However, in view of two aspects, it makes sense to include replication strategies: (i) edge cloudlets are typically more unreliable compared to Cloud Computing infrastructures, and (ii) user mobility requires data migration in order to maintain the benefits of proximate retrieval.

CDN integration | This chapter has explored the data flow of a reverse CDN, i.e., data flows from end devices that capture data towards storage locations. In conjunction with replication, future work needs to include CDN functionality in the framework, e.g., mechanisms for the end devices to efficiently retrieve data.

Automatic rule generation | Rules in our framework are either created by the end user or pushed globally to the devices. Creating sensible rules requires the anticipation of future usage and request patterns (e.g., where and when data will be requested). For future work, we envision automatic rule generation. Especially techniques in the domain of machine learning could contribute to this, e.g., by learning from a large body of usage patterns.

[9] https://github.com/Telecooperation/vstore-framework (accessed: 2020-02-12)

[10] https://github.com/Telecooperation/vstore-master (accessed: 2020-02-12)

[11] https://github.com/Telecooperation/vstore-node (accessed: 2020-02-12)

[12] https://github.com/Telecooperation/vstore-android-filebox (accessed: 2020-02-12)

CHAPTER 10

Microservice Adaptations

Chapter Outline

10.1 Introduction

The previous two chapters have investigated strategies for the placement of functional application parts (Chapter 8) and data (Chapter 9). The concepts presented in those two chapters can be leveraged to adapt an Edge Computing system at runtime, e.g., by re-placing parts of applications or changing the storage location of data. This chapter introduces another dimension of adaptation, building on *flexEdge*, our concept for an Edge Computing execution framework that was introduced in Chapter 7. In *flexEdge*, individual parts of applications are realized as microservices, and can be composed into more complex services, forming a processing chain of subsequent microservices. This is also the predominant execution model, e.g., in Serverless Computing (see Section 7.2.3).

While we are able to adapt the *management* of the services, e.g., through the placement of services (see Chapter 8), the services *themselves* and their internal functioning remains non-adaptable. More specifically, microservices are implemented to deliver a functionality in one particular way and cannot vary *how* the functionality is provided, e.g., by providing different *variants* of a microservice. Those variants may, for instance, differ in the algorithms they use to perform a task. As another example, some services need additional auxiliary data, which can also be varied (e.g., by using different pre-trained models for machine learning applications).

Overview of concept. Different variants of a microservice potentially have an impact on two metrics: (i) the computational complexity, reflected in the execution time and resource demand of a request, and (ii) the quality of result (QoR). The latter can be defined and measured in different ways, e.g., by the accuracy of the result, i.e., its deviation from a (numeric) optimum, or by the (subjective) perception of a user. Overall, these two metrics form a tradeoff, in the sense that more accurate results typically require more computational effort, which leads to higher execution times and/or increased resource demands. On the other hand, if we are willing to sacrifice computational quality, we can perform the same tasks with fewer resources.

This observation is especially remarkable in the context of Edge Computing, if we recall some of its characteristics. On the one hand, computing resources available in Edge Computing are less powerful compared to their cloud counterparts, making efficient computing an important requirement to cope with scarce resources. Similarly, achieving resource elasticity is more challenging in Edge Computing, since the total available resources at a given location are much more limited. On the other hand, many edge applications have stringent requirements on the overall latency. At the same time, such mission-critical applications can be flexible regarding the quality of the computation result. Examples can be found in the domain of image or video processing, and for recognition tasks. To illustrate the practical impact of inaccurate computations, Chippa et al. [Chi+13] surveyed different kinds of applications and found that, on average, applications spent 83 % of their runtime on computations that are error-tolerant.

Current Edge Computing frameworks, however, do not consider this tradeoff between computation effort and the quality of the computation result, and hence, miss out on this optimization opportunity. In this chapter, we present the novel concept of *adaptable microservices*. We re-define microservices as blueprints for the delivery of a particular functionality that can be adapted w.r.t. (i) the algorithms they use to perform a task, (ii) parameters, and (iii) auxiliary data required for the computation. The possible *variants* are implemented within the program code of a microservice and can be selected upon its instantiation. Additionally, through a control channel, the current variant of a service instance can be changed at runtime. The selection of the specific variant can be made according to certain requirements, e.g., a maximum tolerable execution time or a minimum quality of result. Furthermore, by having service variants with varying resource requirements, service variants are a way to bring the much-valued resource elasticity of Cloud Computing to the domain of Edge Computing. Service variations are applied to an individual microservice, but they also have to be considered in the context of a microservice chain. For example, changing a variant of one microservice might have a disproportionate impact on the overall quality or execution time of the entire service chain.

We propose to include this concept of adaptable microservices in an Edge Computing framework. In such a system, clients would submit an abstract definition of the desired microservice or service chain with their individual requirements regarding execution time and QoR to a controller, which would in turn have to make the following decisions: (i) which service variant to choose for instantiation in each step of the chain, and (ii) the assignment of user requests to service instances (since multiple services in different variants might be available). Furthermore, the controller might choose to change the variant of a particular microservice at runtime, e.g., switch the algorithm with which the service performs its task. Especially in cases where microservice instances are shared between multiple microservice chains, this becomes a non-trivial optimization problem, because users that share (parts of) a chain might have conflicting optimization goals.

Summary of contributions. This chapter is intended to open up another dimension of adaptability in Edge Computing and presents an initial concept and case study for *adaptable microservices*. In summary, the contributions of this chapter are threefold:

- We propose the concept of *adaptable microservices* in the context of an Edge Computing environment (Section 10.3). To do so, we revise our previously proposed concept of microservices. We define the adaptability of microservices in three dimensions (algorithms, parameters, and auxiliary data).
- In a first explorative study, we demonstrate the practical impact of service variants using representative examples. First, we study how accurately we can profile the execution times of service variants, using different hardware setups and features (Section 10.4.4). Second, we evaluate the impact of the variants for single microservices and for service chains (Section 10.4.5).
- We present a concept for the integration of adaptable microservices into an existing Edge Computing framework (Section 10.5), detailing several components required for the orchestration of those service variants across edge surrogates and users.

10.2 Background and Related Work

Our contribution explores microservice adaptations in the context of Edge Computing. Some previous works explore the adaptation of services in other contexts (Section 10.2.1). We also review related work in the domain of approximate computing (Section 10.2.2).

10.2.1 Service Adaptation

Adaptation of services has been explored in the context of service-oriented architectures (SOA) and Web Services (WS) [Pap03]. Chang et al. [CLK07] present a survey of common adaptation methods in service-oriented computing. Hirschfeld and Kawamura [HK06] define adaptability in three dimensions: *what* (e.g., computation/behavior or communication), *when* (e.g., at compile time or runtime), and *how* (e.g., composition or transformation). Service adaptations can for instance be realized using adaptation templates [Kon+06]. Moser et al. [MRD08] propose to adapt WS-BPEL[1] services. Contrary to our approach, they do not change the internal working of the service but replace one service with another. Besides adapting the services, some have investigated the dynamic selection of adaptation strategies, e.g., in [PS11].

[1] Web Service Business Process Execution Language

From a software engineering point of view, variants of services can be realized using *Software Product Lines* (SPL). SPLs are a development approach for reusable and interchangeable software [McG+02]. SPLs are characterized by their variability [vBS01] and this variability can for instance be represented with *feature models* [BD07]. Olaechea et al. [Ola+12] present a language and tools for the modeling of SPLs, supporting multiple optimization goals. Based on such feature models, Sanchez et al. [SMR13] present a heuristic-based method for the selection of an optimal configuration. Dynamic software product lines are capable of adapting, e.g., to user requirements or resource constraints [Hal+08]. As an example, Weckesser et al. [Wec+18] examine the reconfiguration of dynamic software product lines. Reconfiguration is done based on consistency properties and learned performance-influence models. The authors however do not consider service chains, and hence, cannot capture the interdependencies of adapting multiple services in a service chain.

More recent works have proposed adaptations for microservices and in the context of the IoT. Kannan et al. [Kan+19] present *GrandSLAM*, a microservice execution framework aimed at maximizing the throughput and reducing SLA[2] violations. They do not modify the microservices themselves but instead change the request distributions by (i) reordering requests and (ii) batching requests. It is worth noting that these techniques can be used in conjunction with our proposed approach. Mendonça et al. [Men+18] discuss the tradeoff between generality and reusability in self-adaptive microservices. Bhattacharya and De [BD17] survey adaptation techniques in computation offloading, considering only the degree of concurrency and workload heterogeneity as variations in the applications. Some works present adaptation models for specific applications, e.g., streaming analytics [Zha+18a], or to realize fault tolerance [Zho+15]. Others adapt the granularity of the services and not the underlying functionalities [HB16]. In contrast, we present a general concept for the adaptation of the internal functioning of microservices.

10.2.2 Approximate Computing

Approximate computing trades computation quality for a reduction in the required effort to perform that computation [Mit16]. The motivation to use approximate computing stems from the fact that in many problem domains of science and engineering, exact results are not required, but only results that are *good enough*. Examples can be found in the domain of digital signal processing, multimedia, and data analytics. Besides algorithmic resilience, *users* are also tolerant of inaccurate results. Examples are search results in information retrieval or the quality of images and video streams. In addition, the usage context might also influence the required computation quality [MFP20].

General related work & surveys. At the top level, we can distinguish between hardware and software approaches for approximate computing [Mor+18b]. Xu et al. [XMK16] classify approximate computing approaches into three layers: (i) program, (ii) architecture, and (iii) circuit. According to this classification, our approach of adaptable microservices falls into the first category. Moreau et al. [Mor+18b] present a taxonomy for approximate computing techniques. Besides the distinction between hardware and software techniques, the authors suggest

[2] Service Level Agreement

classifying techniques according to (i) architectural visibility, (ii) determinism, and (iii) granularity. The survey from Mittal [Mit16] focuses only on the hardware aspect of approximate computing. The author reviews approximate computing techniques for different processing units (e.g., CPU, GPU, or FPGA) and possible applications. In addition, examples of quality metrics for different applications are presented. Regarding savings from approximate computing, related works consider both the aspects of energy-saving [HO13; MSC15] and a reduction in the execution time [Agr+16]. Gao et al. [Gao+17] show how approximate computing is beneficial on the hardware level in terms of saving resources and increasing security. As for energy savings, Moreau et al. [MSC15] note that battery technology is advancing slowly for mobile devices. This is in line with one of the main reasons for performing Edge Computing (see Section 3.1), i.e., saving energy on mobile devices.

Hardware-level approximate computing. Hardware approaches work by introducing imprecise logic components [Gup+11; Ye+13] or using techniques like voltage overscaling [Moh+11]. In Edge Computing, we cannot implement approximate computing on a hardware level, given that we opportunistically leverage existing, heterogeneous devices over which we have no direct control. Hence, we need to move the concept of approximate computing to the software layer, i.e., modifying the way that applications work.

Application-level approximate computing. One example for application-level approximate computing is *FoggyCache* [Guo+18]. The authors propose to reuse computation results across devices, based on the observation that similar contextual properties map to the same or similar outcome. Perez et al. [PBC17] have examined the latency-accuracy tradeoff in MapReduce jobs when applying approximate computing. Chippa et al. [Chi+13] conduct a study in which they analyze the resilience of different applications to result-inaccuracies. As demonstrated in [Agr+16], different approximate computing techniques can be combined. The authors use loop perforation, reduced precision computation, and relaxed synchronization on applications from the domains of digital signal processing, robotics, and machine learning. Their results suggest that up to 50 % in execution time can be saved while producing acceptable results.

Programming frameworks. Park et al. [Par+14] present a framework for approximate programming. Developers can specify accuracy constraints through annotations and based on those, the framework automatically identifies operations that are safe to approximate. Extensions to programming languages that support specific data types to represent the characteristics of approximate computing have been introduced in [BMM14] and [Sam+11]. Other works have demonstrated the potential impact of approximate computing in different application domains, e.g., iterative methods [Zha+14], image compression [AKL18], rendering [WD10], artificial neural networks [Zha+15b], and deep learning [Che+18a].

Edge- & IoT-related. Few previous works exist that apply approximate computing to domains that are related to Edge Computing. Zamari et al. [Zam+17] combine approximate computing with Edge Computing in an IoT scenario where sensor data is to be sent to the cloud for analytics. In-transit edge nodes contribute to the analytics by carrying out intermediate computations. This is coupled with approximate computing techniques on a software level, such as reducing the number of iterations or skipping certain parameter values. Wen et al. [Wen+18] employ a similar approach. They present *ApproxIoT*, combining approximate computing (by using only samples of a raw data stream) with hierarchical processing. Schäfer et al. [Sch+16a] introduce several metrics for the *quality of computation* (QoC), for example, speed, precision, reliability, costs, and energy. They extend their *Tasklet* system [Edi+17]—an offloading middleware for distributed computing—to provide execution guarantees w.r.t. these QoC metrics. Compared to our adaptations, they do so not by modifying the internal functioning of the computation unit but by controlling their distribution across the surrogates. For example, constraints are applied on which machines tasklets can be executed, and computations are carried out in parallel or redundantly. Another difference to our approach is that tasklets are more fine-grained, in the sense that they consist of application subroutines instead of entire services.

Microservices & mobile devices. Gholami et al. [Gho+19] propose the usage of different versions of a microservice (lightweight or heavyweight), primarily for the purpose of scaling the application. They demonstrate how this multi-versioning can be included in Docker containers. In a broader context, Pejovic et al. [Pej18] outline the challenges for approximate computing on mobile devices with a focus on the users' needs. Similarly, Machidon et al. [MFP20] have noted that the field of approximate computing for mobile devices still lags behind its counterparts in the desktop and server environment. Using the example of mobile video decoding, the authors demonstrate how the acceptable quality degradation can vary according to the user's current context.

Summary. Table 10.1 summarizes the landscape of related work in approximate computing by listing representative examples. The bottom row of the table also shows how our approach compares to the existing ones. Compared to previous approaches, we propose three *general* adaptations to the *internal* functioning of application components. The tradeoff between result quality and execution time is especially relevant in Edge Computing applications such as image processing or recognition tasks. While some works have investigated the automatic selection of variants in the context of SPLs [SMR13], this has not been investigated in the context of Edge Computing. In Section 10.5, we will present a concept for the integration of such a variant selection in an Edge Computing framework.

10.3 Adaptable Microservices for Edge Computing

In the remainder of this chapter, we use the term *service adaptation* to refer to the *internal functioning* of the services. This definition stems from the observation that a particular functionality can be implemented in different ways, leading to many possible *service variants* between which we can switch at runtime. This adaptation is orthogonal to other runtime optimizations that can be taken to provide certain guarantees, e.g., the scaling of microservices or their migration. Compared to costly migration and (re-)placement strategies, we believe that our approach is a sensible alternative because it allows for a quick reconfiguration of instance variants and, therefore, service instances can be kept active for a longer period of time.

Reference	Type of approach	Description	Focus
[Gup+11; Ye+13]	hardware / logic design	usage of imprecise logic components	savings in energy and required area
[Moh+11]	hardware / power management	performing voltage overscaling	tradeoff between energy consumption and error rates
[Par+14]	software / framework	determining operations that are safe to approximate, based on specified accuracy constraints	energy savings and reduction in programmer effort
[Guo+18]	software / general	reuse of cached computation results across devices	reduction in latency and energy consumption
[Zam+17; Wen+18]	software / application-specific	combine approximate computing with edge analytics of IoT data	speedup-accuracy tradeoff
[Sch+16a]	software / general	controlling the distribution of computing units across surrogates	providing quality of computation (QoC) guarantees
[Gho+19]	software / general	multi-versioning of software with different Docker containers	adapt software to different performance requirements
[MFP20]	software / application-specific	downsampling of video resolution on mobile devices	energy savings with acceptable quality degradation
Section 10.3	software / general	provide microservice adaptability in three dimensions	tradeoff between result quality and execution time

TABLE 10.1: EXAMPLES OF APPROACHES FOR APPROXIMATE COMPUTING

Contrary to previous approaches, our concept of adaptable microservices combines the following three characteristics: (i) we adapt the internal functioning of a microservice, i.e., we operate on the application level and adaptations are implemented in the program code of the microservices, (ii) we propose adaptations in three general dimensions, and (iii) we envision a control entity that automatically selects and changes the service variants at runtime. This section focuses on the first two characteristics and presents our conceptual model. Section 10.5 details the integration into an Edge Computing framework that controls the variant execution.

We propose to make microservices adaptable in the following three dimensions:

(i) Algorithms | A task can typically be performed by a variety of algorithms. Those not only differ in their runtime complexity, and hence, result in varying execution time, hardware requirements, and energy consumption, but also in their suitability for different applications. Taking the example of compressing an image, some compression algorithms are better suited for photographs while others perform better on vector graphics.

(ii) Parameters | Parameters are variable inputs to the microservice that influence its execution behavior. We model parameters as key-value pairs. Parameters can, for example, customize the algorithm that is used. Taking the same example of image compression, the desired image quality would be a parameter for such a microservice. Parameters can also be used to explicitly limit the execution time of a microservice, e.g., via loop perforation[3] [Sid+11].

(iii) Auxiliary Data | Some algorithms require auxiliary data to function. This data is often retrieved from external sources. An example in the domain of machine learning are pre-trained models. This auxiliary data can also influence the execution time and the computation result. For example, in recognition tasks performed by neural networks, more complex models produce more accurate results, but require more computing resources or take longer to complete the task.

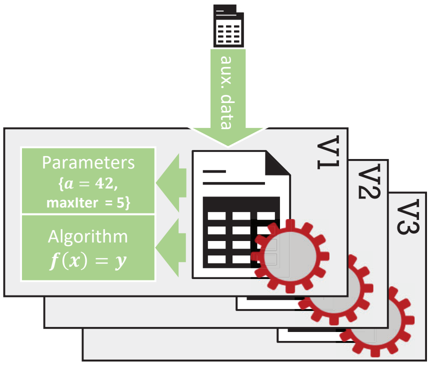


Figure 10.1: Variants of adaptable microservices

Figure 10.1 visualizes the concept of adaptable microservices. A given service might be variable in one or more of these dimensions. We define the possible

[3] loop perforation refers to skipping certain iterations in a loop or breaking the loop after a number of iterations.

combinations of all three adaptation dimensions as *service variants* (denoted with $V1, V2, V3$ in the figure).

> DEFINITION 10.1: SERVICE VARIANTS
>
> Given a set of implemented algorithms $\mathscr{A}$, parameters $\mathscr{P}$, and auxiliary data $\mathscr{D}$ for a microservice, a *service variant* Var_M of a microservice M is defined as $Var_M \subseteq \mathscr{A} \times \mathscr{P} \times \mathscr{D}$ with $p_i = v_i, i = 1 \dots n$ as values for the parameters. Note that $\mathscr{A}$ and $\mathscr{D}$ are finite sets, whereas $\mathscr{P}$ typically is an uncountable set, e.g., in case the parameters contain real numbers.

We assume that there are no variants across a service chain that are mutually exclusive. Should one want to consider this case, constraint solvers can be for selecting valid variants [BSC10]. We further assume that each of the variants is implemented in the microservice. For example, if a microservice can be implemented using different algorithms, all those algorithms are included in the source code of the service. At any given time, a service maps its current variant to an internal state that determines how it is executed when requests are processed.

The different service variants impact the result of the computation in two ways. First, the execution time varies, e.g., when less complex algorithms are invoked or loop iterations are skipped. Naturally, this leads to a reduced energy consumption of the surrogate which executes the microservice. Second, service variants impact the quality of result (QoR). Depending on the application, QoR needs to be defined differently. We can divide QoR-metrics into two categories: (i) user-centered and (ii) numeric. For user-centered metrics, techniques like questionnaires or focus groups can be used to assess the perceived quality of result. Note that this might not only vary from one user to another but also might depend on the usage context (as noted in [MFP20]). As a numeric metric, we can for example quantify the error in the computation, i.e., the deviation from a numeric optimum or the accuracy of the result.

10.4 Case Study

In a first case study, we build a test environment to demonstrate our concept (Section 10.4.1). We implement six different microservices (Section 10.4.2) that are adaptable in different aspects, and with those, we construct two microservices chains (Section 10.4.3). First, we study how accurately we can construct a model to estimate the execution time of those services (Section 10.4.4). The results also allow us to identify which features (e.g., hardware characteristics) are the most relevant for such a model. Then, we demonstrate the practical impact of the different service variants on the execution time and (where applicable) QoR (Section 10.4.5).

10.4.1 Implementation and Test Environment

For the implementation, we partly rely on concepts and components of flexEdge (see Chapter 7). The microservices follow the design that was presented in Section 7.3.1. Microservices are executed on the *agents* of flexEdge (see Section 7.4.3). We implement the possibility to select the microservice variant at the start of the service by passing arguments to the Docker CLI that are then parsed by the service at its start. For the case study, we manually select the microservice variant to study its impact and do not rely on the flexEdge controller. Future work will incorporate the specification of user constraints and the automatic selection and adaptation of service variants by the controller (see Section 10.6).

In addition to the queue that holds requests, each microservice instance is extended with a *control queue*, which also uses *RabbitMQ* as a message broker. Messages are published to this queue to trigger a change in a service variant. Similarly, through a REST-style API, the microservices can be queried to retrieve which variant is currently active. To provide this interface, each microservice implements an abstract *Microservice Manager* class. This implementation is then connected to a listener for the control queue in order to parse incoming requests. As an example, Figure J.1 in Appendix J shows a UML class diagram for this functionality of the face detection microservice.

We use a similar setup as described in Section 7.6.1. The controller and microservice store run on the same hardware as described there. We also use the same Lenovo *ThinkCentre M920X Tiny* as one edge agent node. In addition, for different experiments we use different AWS EC2 virtual machines. Requests are issued from a computer in the same local network as the controller, agent, and microservice store.

10.4.2 Microservices

We implement the following adaptable microservices, as summarized in Table 10.2:

TABLE 10.2: OVERVIEW OF SERVICE VARIANTS

Microservice	**Variants**		
	Algorithms	Parameters	Auxiliary Data
Face detection	{*LBP-Classifier*, *Haar-Classifier*}	{*scale-factor*, *min-neighbors* }	∅
Object detection	∅	∅	{*faster_rcnn_inception_v2_coco*, *ssd_mobilenet_v1_coco*, *ssd_mobilenet_v1_fpn*, *ssd_mobilenet_v1_ppn*, *ssd_mobilenet_v2_coco*, *ssd_resnet50_v1_fpn*, *ssdlite_mobilenet_v2_coco*}
Image compression	∅	{*compression-quality*}	∅
Image blurring	{*Gaussian blur*, *Median blur*}	{*kernel-size*}	∅
Image upscaling	∅	∅	{*psnr-large*, *psnr-small*,, *noise-cancel*,*gans*}
3D mesh reconstruction	∅	∅	{*meshrcnn*,*pixel2mesh*, *sphereinit*,*voxelrcnn*}

(i) Face detection | This microservice was introduced in Section 7.5.1. Its variants differ in algorithms and parameters. For the algorithms, we can use two different types of cascade classifiers available in OpenCV[4]: (i) *LBP* and (ii) *Haar*. In general, LBP is faster but produces less accurate results. The microservice furthermore expects two parameters: (i) *scale-factor* and (ii) *min-neighbors*. The first parameter determines the scaling between two levels of up- or downscaling (because both algorithms work only on predefined model dimensions). The second parameter *min-neighbors* specifies the minimum number of neighbors for candidate rectangles for those to be retained. Higher values for this parameter lead to fewer faces being detected but at the same time, this also decreases the number of false-positives.

(ii) Object detection | This microservice was also introduced in Section 7.5.1. The microservice uses different auxiliary data with pre-trained tensorflow models[5]. The models differ in their execution speed and mean average precision. In total, we use six different models.

(iii) Image compression | Using the image encoding function of OpenCV, this microservice compresses a given input image using JPEG. JPEG is a lossy compression method. As the only variation, the compression quality can be specified as a parameter.

(iv) Image blurring | Given an input image and an array of rectangular regions, this microservice blurs the given regions of the image. To perform the operation, we use OpenCV's blur function. The blurring can be performed by two different algorithms: (i) *Gaussian blur* and (ii) *median blur*. The Gaussian blur is a linear filter that is faster but does not preserve edges in the original image. In contrast, the median blur is a non-linear filter that is able to preserve edges. For both algorithms, a *kernel size* is used as a parameter to determine the size of the convolution matrix.

(v) Image upscaling | This microservice produces an upscaled image of the input image. It also aims at enhancing the quality of the upscaled image by using Residual Dense Networks (RDN). We use an existing Keras[6]-based implementation[7] as a basis for our microservice. We use four different pre-trained models that are variants of auxiliary data: *psnr-large*, *psnr-small*, *noise-cancel*, and *gans*. Except for the *gans* model (which quadruples the resolution), these models double the original image resolution.

(vi) 3D mesh reconstruction | This microservice aims at reconstructing a 3D mesh representation of an object in a (2D) picture. Gkioxari et al. [GJM19] showed how this can be achieved using convolutional neural networks. We use the author's published code[8] as the basis for our microservice. Four different models are used as auxiliary data. Some reconstruct only the shape of the object while others use voxels to achieve a more realistic representation of the object.

[4] see https://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html (accessed: 2020-04-06)

[5] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md (accessed: 2020-04-22)

[6] https://keras.io/ (accessed: 2020-04-21)

[7] https://github.com/idealo/image-super-resolution (accessed: 2020-04-16)

[8] https://github.com/facebookresearch/meshrcnn (accessed: 2020-04-16)

10.4.3 Microservice Chains

From the microservices described in the previous Section 10.4.2, we construct two service chains. These service chains represent prominent applications in the domains of computer vision, recognition, and machine learning. Service chains with different variants can also represent use cases in which data is pre-processed for a more efficient offloading (termed *offload shaping* [Hu+15a]) of subsequent microservices, e.g., as investigated in [Li+15].

(i) FACE ANONYMIZATION | Given an image as input, this chain anonymizes faces by blurring them. First, the original image is compressed. Afterward, a face detection is performed, outputting detected faces as rectangular coordinates to the next service. As a final step, the image blurring microservices blurs the regions returned by the face detection microservice. An illustrative example of this service chain is shown in Figure 10.2(a).

(ii) 3D MESH RECONSTRUCTION OF UPSCALED IMAGES | This microservice chain first performs an upscaling of an input image and then reconstructs a 3D mesh from the upscaled image. Figure 10.2(b) illustrates an example execution of this chain.

10.4.4 Execution Time Estimation

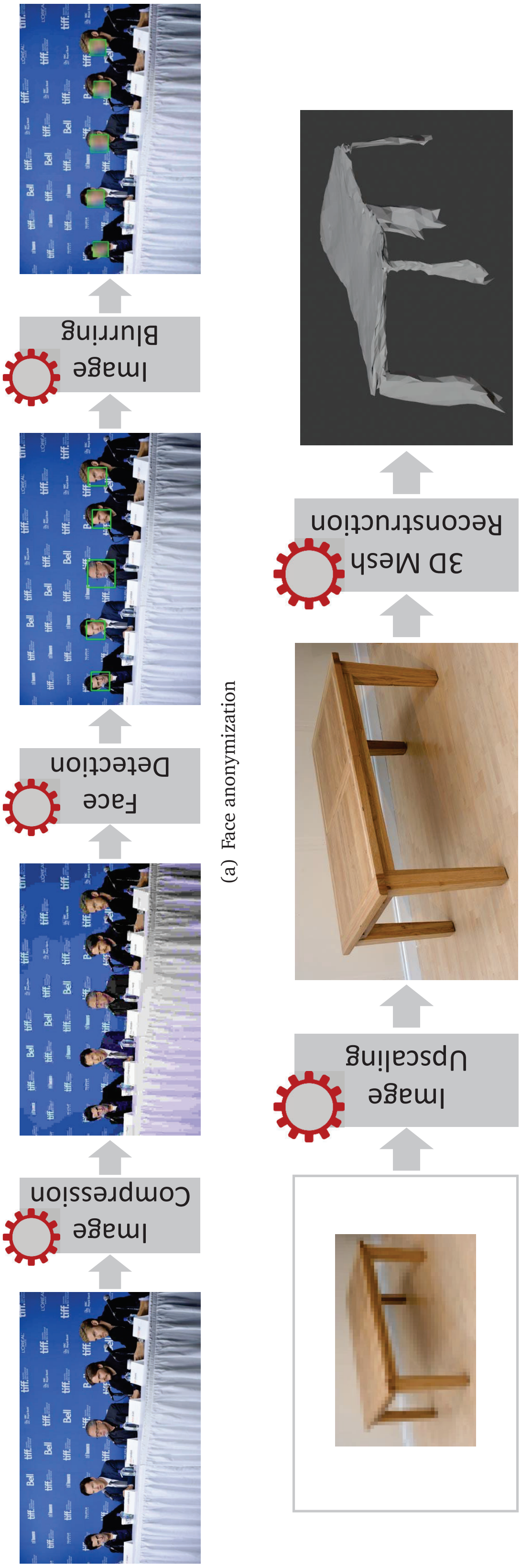
First, we study how accurately we can model the execution time of the microservices and which features are relevant for creating a model of the execution time that is as accurate as possible. In an Edge Computing framework where adaptable microservices are integrated (see Section 10.5), this step would be performed offline and serve as base knowledge for runtime decisions. It would allow, for instance, to estimate the execution time of a request and based on this estimation, make decisions about the placement or user-to-instance assignments.

Methodology and experimental setup. To estimate the execution time of microservices, we build regression models using supervised learning methods. We use three variants of estimators implemented in *scikit-learn*[9], a machine learning library for Python: (i) decision trees regressor, (ii) random forest regressor, and (iii) extra tree regressor. For each estimated model, the R^2*-score* is computed to assess its quality. This metric gives an indication of how accurate the model is.

To analyze the impact of different underlying hardware configurations, we run this evaluation on different AWS EC2 instances types as summarized in Table 10.3. They differ in CPU and memory configuration and are optimized for either general-purpose, computing, or memory-intensive applications. For each instance type, we run benchmarks of the face detection and object detection microservice. During the execution of the microservices, we also record statistics on available hardware resources and system load. Those will serve as possible features to build a model of the execution time (for instance, to be able to predict the execution time given different load levels on the system).

For the face detection, we use the two different face detection algorithms, *LBP* and *Haar* [Kad+14]. The *scale-factor* parameter is varied from 1.0 to 1.9 in 0.1 increments and values for *min-neighbors* are varied from 1 to 10. We use 46 160

[9] https://scikit-learn.org (accessed: 2020-04-13)



(a) Face anonymization

(b) 3D mesh reconstruction of upscaled images

FIGURE 10.2: MICROSERVICE CHAINS USED FOR THE EVALUATION

TABLE 10.3: INSTANCE TYPES USED FOR BENCHMARKING

Type	vCPUs	Clock rate	Memory	Remarks
t2.micro	1	2.5 GHz[a]	1 GB	general-purpose
t2.small	1	2.5 GHz[a]	2 GB	general-purpose
t2.medium	2	2.3 GHz[b]	4 GB	general-purpose
t2.large	2	2.3 GHz[b]	8 GB	general-purpose
t2.xlarge	4	2.3 GHz[b]	16 GB	general-purpose
t2.2xlarge	8	2.3 GHz[b]	32 GB	general-purpose
c5.large	2	3.4 GHz[c]	4 GB	compute-optimized
c5.xlarge	4	3.4 GHz[c]	8 GB	compute-optimized
c5.4xlarge	16	3.4 GHz[c]	32 GB	compute-optimized
r5d.large	2	3.1 GHz[d]	16 GB	memory-optimized
r5d.2xlarge	8	3.1 GHz[d]	64 GB	memory-optimized

[a] Intel Xeon Family

[b] Intel Broadwell E5-2695v4

[c] Intel Xeon Platinum 8124M

[d] Intel Xeon Platinum 8175

different images from the *WIDER FACE*[10] dataset. For the object detection, we use two different models (*ssd_mobilenet_v1_coco* and *faster_rcnn_inception_v2_coco*) on the *val2017* dataset included in the *Coco Dateset*[11].

For each instance type, we list the combination of features and regression methods that lead to the highest R^2-score. Features can be properties related to the variant of the microservice (e.g., a parameter) or attributes of the machine where it is executed. Table 10.4 shows the results for the face detection and Table 10.5 the results for the object detection. In the table, the features are ordered in decreasing order of importance, i.e., to what extent they contribute to the prediction of the execution time.

Impact of the machine types. From the results, we can observe that especially with the more powerful machines, we can achieve high R^2-scores and, hence, a high accuracy of the model. For less powerful types of machines, e.g., the *t2.micro* and *t2.small*, we get much lower scores. This is likely due to a greater variance in execution times that happens because *t2*-type instances are so-called *burstable instances*, i.e., if the system is overloaded, the CPU performance of the virtual machine is temporarily increased. Since this is likely to happen with the least powerful types we used, the high variance of execution times is due to the constant on-off switching of the performance boost.

Differences in estimators and features. As a second observation, in all but one case (face detection on a *c5.4xlarge* instance), the *extra tree* regressor led to the highest R^2-score. We can also observe great differences in the most relevant features for the execution time estimation. These differences can both be seen within

[10] http://shuoyang1213.me/WIDERFACE/ (accessed: 2020-04-25)

[11] http://cocodataset.org/ (accessed: 2020-04-25)

Table 10.4: Execution time estimation result for the face detection microservice

Instance	Regressor	Features[a]	R^2-Score
t2.micro	Extra Tree	SF, DF-A, MN	0.4675
t2.small	Extra Tree	CF, DF-A, MEM-A	0.4317
t2.medium	Extra Tree	CF, CPU-U, DF-C	0.8717
t2.large	Extra Tree	CPU-U, CF, SF, NF	0.9456
t2.xlarge	Extra Tree	CPU-U, CF, SF, MN, NF	0.9842
t2.2xlarge	Extra Tree	CF, SF, CPU-U, NF	0.9856
c5.large	Extra Tree	CF, SF, CPU-U, NF	0.9887
c5.xlarge	Extra Tree	CF, SF, CPU-U, NF	0.9914
c5.4xlarge	Decision Tree	CF, CPU-U, SF, CPU-F, MEM-U, DF-C, MEM-A	0.9965
r5d.large	Extra Tree	CF, SF, CPU-U, NF	0.9883
r5d.2xlarge	Extra Tree	CF, SF, CPU-U, NF	0.9860

[a] **CF:** classifier, **CPU-F:** CPU frequency, **CPU-U:** CPU usage, **DF-A:** detected faces (absolute number), **DF-C** detected faces (correct percentage), **MEM-A** available memory, **MEM-U:** used memory, **MN** min-neighbors, **NF** number of faces, **SF:** scale-factor

Table 10.5: Execution time estimation result for the object detection microservice

Instance	Regressor	Features[a]	R^2-Score
t2.micro	n.a[b]		
t2.small	Extra Tree	M, CPU-U, MEM-T	0.5651
t2.medium	Extra Tree	MEM-AP, M, MEM-U, C, CPU-F	0.6938
t2.large	Extra Tree	M, MEM-A, MEM-U	0.6827
t2.xlarge	Extra Tree	M, MEM-A, MEM-T	0.6412
t2.2xlarge	Extra Tree	M, MEM-AP, CPU-U	0.7690
c5.large	Extra Tree	M, CPU-U, MEM-T	0.9970
c5.xlarge	Extra Tree	M, CPU-U, MEM-T	0.9969
c5.4xlarge	Extra Tree	M, CPU-U, MEM-T	0.9968
r5d.large	Extra Tree	M, CPU-U, MEM-T	0.9968
r5d.2xlarge	Extra Tree	M, CPU-U, MEM-T	0.9970

[a] **C:** correctness, **CPU-F:** CPU frequency, **CPU-U:** CPU usage, **M:** model, **MEM-AP:** available memory (percentage), **MEM-A** available memory (absolute), **MEM-T** total memory, **MEM-U:** used memory

[b] hardware configuration not sufficient to run the microservice

one microservice, depending on the instance type, and across microservices. The estimation for the face detection mostly used the classification algorithm as the most relevant feature. With more powerful hardware, the classifier, the *scale-factor* parameter, and the current CPU usage are consistently ranked the most relevant features, while for less powerful machines, the *min-neighbors* parameter and the number of detected faces were included in the features.

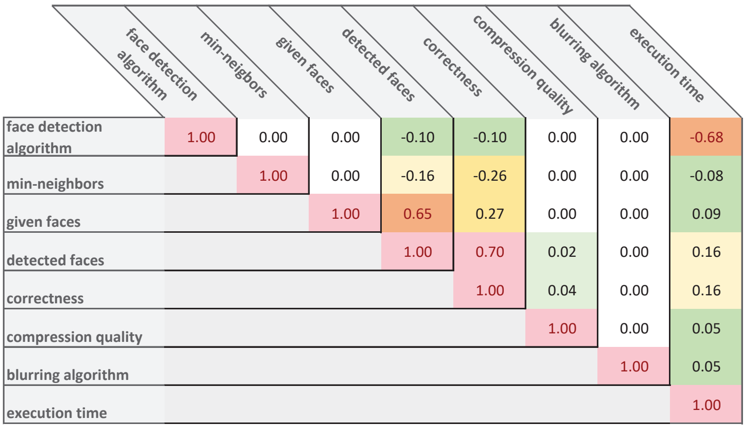
Contrary to the face detection microservices, for the object detection, we can see a clearer division of relevant features depending on the instance type. While for *t2*-type instances, the available memory is always a highly ranked feature (except for the *t2.small* instance), this changes in favor of the CPU utilization for *c*-type and *r*-type machines. Another difference is that the *t2*-type instances lead to significantly lower accuracies of the model, as shown by the R^2-score.

Summary. In summary, this analysis of execution time estimators has shown that we are able to accurately profile the different variants of microservice. This is an important building block for the selection and adaptation of suitable microservice variants at runtime. However, we could also observe that this estimation has to be tuned to the individual microservice w.r.t. the selection of the hardware and features that are used for the estimation. This further motivates our design presented in Section 10.5, in which the offline model is periodically updated with runtime statistics from the execution environment.

10.4.5 Impact of Service Variants

We now study the impact of the different microservice variants. To do so, we measure the correlation between different variables that relate to the service variants and the outcomes of the computations. Most importantly, we want to assess the change in execution time. In addition, for the face detection algorithm and, consequently, for the face anonymization service chain, we also analyze the impact on the quality of the result.

To measure the pairwise correlation between variables, we use the *Kendall rank correlation coefficient* throughout this section. Contrary to other metrics for correlation, such as the *Pearson correlation coefficient*, it has the advantage that it does not assume a linear relationship between variables.



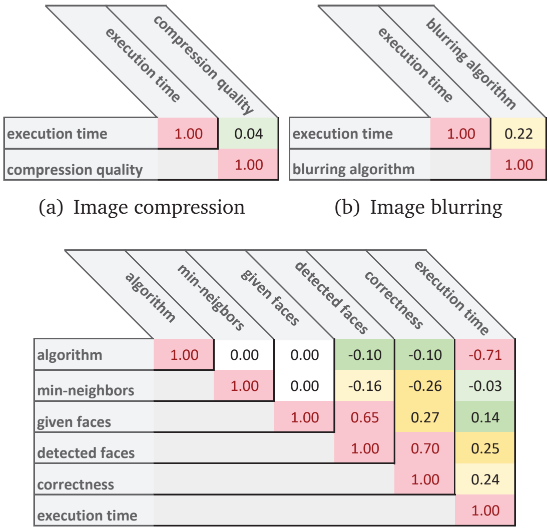
	face detection algorithm	min-neighbors	given faces	detected faces	correctness	compression quality	blurring algorithm	execution time
face detection algorithm	1.00	0.00	0.00	-0.10	-0.10	0.00	0.00	-0.68
min-neighbors		1.00	0.00	-0.16	-0.26	0.00	0.00	-0.08
given faces			1.00	0.65	0.27	0.00	0.00	0.09
detected faces				1.00	0.70	0.02	0.00	0.16
correctness					1.00	0.04	0.00	0.16
compression quality						1.00	0.00	0.05
blurring algorithm							1.00	0.05
execution time								1.00

FIGURE 10.3: FACE BLURRING CHAIN: CORRELATION MATRIX OF VARIANTS

Face anonymization service chain. For the first chain, we vary the image compression quality from 1–99 (in steps of 1). We use the two face detection algorithms as described before. The *scale-factor* parameter is set to a constant 1.2, and *min-neighbors* are varied from 0–9 (step size 1). For the final step, the blurring microservice, we use *gaussian blur* and *median blur* algorithms with a fixed *kernel size* of $(23, 23)$. We select 21 images and manually label the correct positions of the faces. Hence, with a small degree of tolerance, besides the absolute number of detected faces, we can also compute a *correctness* value that serves as a metric for the QoR. For each image and combination, we executed the chain five times and averaged the results.

Figure 10.3 shows the correlation matrix of the entire chain. We can observe that the highest correlation value is attained among the face detection algorithm and the execution time. To map this correlation to concrete numbers, on average, the execution time using the Haar classifier was 0.13 s, while for the LBP classifier it averaged to 0.08 s. This means that by changing the variant of the algorithm, we could achieve a reduction in the execution time of 38.46 %. However, this reduction in execution time came at the cost of a reduced correctness value, which dropped from 0.67 to 0.57 on average (-14.92 %). This provides a good example of the tradeoff between the computation complexity (represented by the execution time) and the quality of result (represented by the correct recognition of faces) that is possible to adapt with different microservice variants.

Compared to the face detection algorithm, other variables related to the variants, i.e., *min-neighbors*, *compression quality*, and *blurring algorithm* correlate with the execution time with values of -0.08, 0.05, and 0.05, respectively. It is worth noticing that *min-neighbors* has a much more significant impact on the correctness (with a correlation value of -0.26) than on the execution time.



	execution time	compression quality
execution time	1.00	0.04
compression quality		1.00

(a) Image compression

	execution time	blurring algorithm
execution time	1.00	0.22
blurring algorithm		1.00

(b) Image blurring

	algorithm	min-neigbors	given faces	detected faces	correctness	execution time
algorithm	1.00	0.00	0.00	-0.10	-0.10	-0.71
min-neighbors		1.00	0.00	-0.16	-0.26	-0.03
given faces			1.00	0.65	0.27	0.14
detected faces				1.00	0.70	0.25
correctness					1.00	0.24
execution time						1.00

(c) Face detection

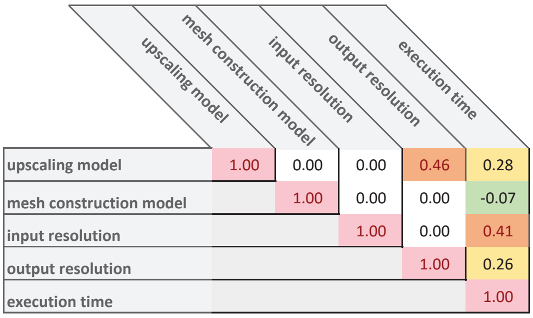
FIGURE 10.4: CORRELATION MATRICES FOR THE INDIVIDUAL SERVICE VARIANTS OF THE FACE ANONYMIZATION CHAIN

We also provide the correlation matrices of the individual services of this chain in

Figure 10.4. Comparing Figure 10.4 with Figure 10.3 demonstrates the difference in correlation of a single microservice versus when this microservice is integrated into a chain. As an example, when executed alone, the blurring algorithm has a correlation value of 0.22 with the execution time but in the entire chain, this value drops to 0.05. A similar change in the correlation score can be observed for the face detection algorithm (-0.68 to -0.71).

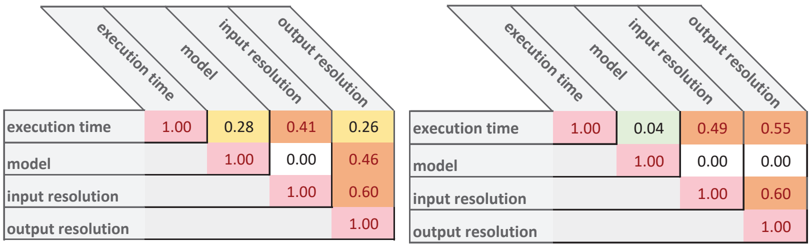
3D mesh reconstruction of upscaled images. For both the image upscaling and 3D mesh reconstruction microservice, we use the four different variants as listed in Table 10.2. As input data, we used 5 images from a dataset depicting furniture[12]. Because the mesh reconstruction microservices offers GPU support, we execute this service chain on an AWS EC2 *p2.xlarge* instance (Xeon E5-2686 v4, 61 GB RAM, Nvidia K80 GPU).

Figure 10.5 shows the correlation matrix for the entire chain and Figure 10.6 the matrices for the individual microservices. Note that for this microservice chain, we leave the exploration of suitable QoR-metrics for future work and focus on the execution times.



	upscaling model	mesh construction model	input resolution	output resolution	execution time
upscaling model	1.00	0.00	0.00	0.46	0.28
mesh construction model		1.00	0.00	0.00	-0.07
input resolution			1.00	0.00	0.41
output resolution				1.00	0.26
execution time					1.00

FIGURE 10.5: MESH RECONSTRUCTION CHAIN: CORRELATION MATRIX OF VARIANTS



	execution time	model	input resolution	output resolution
execution time	1.00	0.28	0.41	0.26
model		1.00	0.00	0.46
input resolution			1.00	0.60
output resolution				1.00

(a) Image upscaling

	execution time	model	input resolution	output resolution
execution time	1.00	0.04	0.49	0.55
model		1.00	0.00	0.00
input resolution			1.00	0.60
output resolution				1.00

(b) 3D mesh reconstruction

FIGURE 10.6: CORRELATION MATRICES FOR THE INDIVIDUAL SERVICE VARIANTS OF THE MESH RECONSTRUCTION CHAIN

The results show that the variants of the upscaling model have more influence than the different mesh reconstruction models (correlation scores of 0.28 and -0.07). As an example, the *psnr-small* model for image upscaling has an average

[12] https://www.kaggle.com/akkithetechie/furniture-detector/data (accessed: 2020-04-24)

execution time of 11.55 s while the *psnr-large* model averages to 58.94 s. The mean values for the *noise-cancel* and *gans* models are 63.93 s and 36.36 s, respectively. This means that by selecting another variant of an image upscaling model, we can reduce the execution time up to 81.93 %. In comparison, the differences for the average execution times of the mesh construction models are smaller (41.26 s for *meshrcnn*, 42.01 s for *pixel2mesh*, 43.12 s for *sphereint*, and 44.38 s for *voxelrcnn*). Hence, here the maximum difference in execution time only amounts to 7.03 %. Naturally, there is also a strong correlation (0.41 and 0.26) of the execution time with the input and output resolution of the upscaled images.

10.5 Integration into an Edge Computing Framework

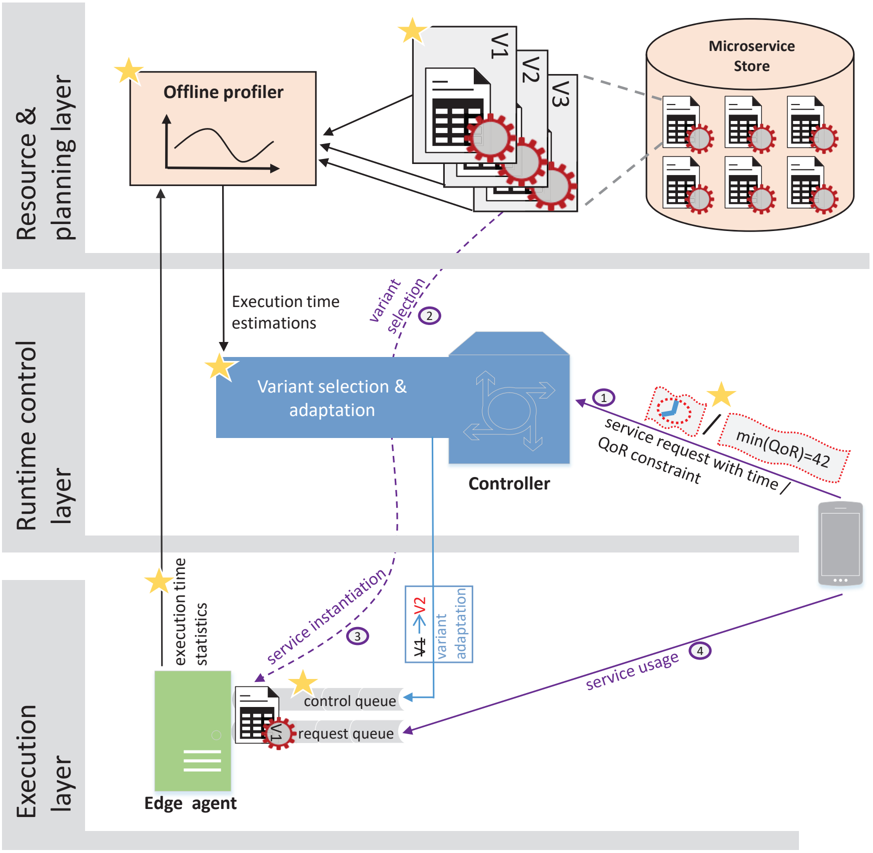


FIGURE 10.7: INTEGRATION OF ADAPTABLE MICROSERVICES INTO AN EDGE COMPUTING FRAMEWORK

Starting from the basis of our microservice-based Edge Computing framework flexEdge (see Chapter 7), this section presents the conceptual design for the integration of service variants into the framework, including new control and monitoring functionalities. Hence, this chapter serves as a blueprint for future work to include our presented concepts into a production-grade Edge Computing system.

Overview of system design. Figure 10.7 shows an overview of our proposed integration concept. Components that are additions to the original flexEdge framework are marked with a yellow star. We structure the design of our system into three layers: (i) a resource and planning layer that provides the adaptable microservices and profiling of the services, (ii) the runtime control layer that manages the variants of the services, and (iii) the execution layer, where the service variants run on the edge agents.

Profiling and monitoring of adaptive services. Variants of a microservice are included in its implementation. For each variant, an offline profiler creates a model to estimate the execution time, given different input sizes. This information serves as a basis for the controller for choosing suitable variants at the start of a microservice or change the variants of running services. Given the heterogeneity of execution environments, not all possible hardware configurations and runtime characteristics (e.g., the current resource usages on the agents) can be considered. Hence, this information is gradually updated at runtime with collected statistics from the agents.

Changing service variants. During the execution of a microservice, its variant can be changed. This is done through a dedicated *control queue* associated with each microservice instance. As an example, in Figure 10.7, the service variant is changed from *V1* to *V2*. This adaptation at runtime can be done for a number of reasons, e.g., when a constraint on the execution time cannot be met, a service might be instructed by the controller to switch to a variant that produces less accurate but faster results.

Control flow. Users can submit their requests for the execution of a service with a constraint on the execution time or the quality of result (shown as ① in Figure 10.7). Based on these constraints and the information from the profiler, a suitable service variant is selected (step ②), instantiated (step ③), and can then take user requests (step ④). Note that for simplicity reasons, the figure only depicts a single microservice. For service chains, the decision-making process is made across all services in the respective chain. Besides further algorithmic contributions (e.g., w.r.t. microservice placement and user-to-instance assignments), this will require scalable monitoring and control mechanisms (see Section 10.6).

10.6 Conclusion and Outlook

In this chapter, we have revised our previously defined concept of microservices. Based on three properties of Edge Computing and its applications—constrained resources, tight constraints on the execution time, and flexibility regarding the quality of the computations—we proposed the general concept of *adaptable microservices*. Specifically, we defined microservices to be adaptable in three aspects, related to the *internal functioning* of the microservices.

In an initial study, we first studied how accurately we can estimate the execution time of an individual service. This is an important building block for an integrated control system that selects service variants at runtime and assigns users to different service variants.

Adaptable microservices allow trading the quality of computations for lower resource utilization (manifested for example in a reduced execution time). Section 10.4.5 has demonstrated this for prominent real-world use cases in the domain of recognition and computer vision tasks. For such complex tasks, we showed that switching to an other microservice variant can substantially reduce the execution time. Alternatively, the less complex variant could be run on less powerful hardware. Compared to Cloud Computing, clustering vast amounts of resources in one location is not possible to the same extent in Edge Computing (e.g., because of limited physical space at points of presence). For this reason, elasticity is typically lower in Edge Computing. The proposed concept of microservice variants can help in mitigating this limited elasticity by *adapting the services to the limitations of the execution infrastructure* and not vice versa.

As the next step, control mechanisms for the automatic selection and change of service variants at runtime need to be implemented in our Edge Computing framework *flexEdge*, as conceptualized in Section 10.5. In particular, we identify the following aspects as a roadmap for future work:

HIERARCHICAL MONITORING AND CONTROL | To ensure that application-specific constraints w.r.t. the execution time and result quality are met, the execution of service chains needs to be monitored. Given the highly distributed nature of the surrogates, having only one centralized controller does not meet the scalability requirements of Edge Computing. Hence, we envision hierarchical monitoring and control mechanisms.

VARIANT SELECTION AND ADAPTATION | Based on continuous monitoring of the execution environment, the available resources on the surrogates, and user requirements, future work will investigate strategies for the selection and adaptation of service variants. This is a challenging optimization problem, especially when microservice instances are shared across users that specify different execution constraints.

NETWORK CONTROL LAYER | In a distributed Edge Computing system, not only the resources on the edge nodes and the microservices' complexity influence the execution time but also the network conditions and types of connections between the nodes (e.g., when the microservices of one service chain run on different nodes). Future work should take this into consideration in two aspects: First, fine-grained monitoring of network conditions can help in making runtime decisions for the placement and assignment of microservices. Second, we can extend the control itself to the network layer, e.g., by reserving bandwidth on links or using SDN to control the data flow between edge nodes.

DEFINING AND WEIGHTING MULTIPLE QOR METRICS | As we have noted, the quality of a computation can be defined in different ways. However, the interplay between user-perceived QoR and mathematical metrics for QoR is not well understood yet. Furthermore, it remains unclear how both types of QoR should be weighted if they are part of one service chain.

DEFINING SERVICE VARIANTS THROUGH SPLS | Software product lines allow for a general modeling of application variants. Using this established technique would also make it possible to model more complex dependencies between variants (e.g., when certain combinations of variants are mutually exclusive).

ONLINE REFINEMENT OF EXECUTION TIME MODELS | We have built offline regression models for the execution time estimation of microservices. Such models cannot, however, take into account all possible hardware and input data that will occur once a service is deployed. Therefore, the models should be refined at runtime with statistics that are collected on the edge nodes during the execution of microservices.

Part V

Epilogue

In the last part of this dissertation, we conclude by summarizing our contributions and outlining future research directions.

CHAPTER 11

Conclusion

Chapter Outline

11.1 Summary

Edge Computing is an emerging paradigm that brings storage and processing capabilities closer to the ever-increasing number of (mobile) end devices, sensors, and actuators. It therefore introduces a new middle-tier between end devices and distant Cloud Computing infrastructures. This brings several advantages compared to the state-of-the-art Cloud Computing paradigm, such as a reduced latency or bandwidth savings in the core network. This in turn enables upcoming applications, such as augmented reality, IoT data analytics, and collaborative gaming. One major challenge towards the widespread availability of Edge Computing is the deployment of elastic, proximate computing resources. For two reasons, we envision such resources to be deployed in urban areas first: (i) the proposed applications for edge computing typically involve densely interconnected people and things, and (ii) both Cloud Computing providers and operators of cellular networks are in the process of deploying general-purpose hardware in or close to the access network, naturally preferring regions with a large number of potential customers.

In this thesis, we have presented contributions in the field of *Urban Edge Computing*, i.e., Edge Computing in an urban environment. This environment is characterized by its heterogeneity in different aspects, which we have considered throughout the contributions of this thesis. First, Edge Computing features a variety of devices, data sources, and consumers. Those are connected using different wireless access technologies (e.g., WiFi or cellular networks). Second, a variety of applications require computation and storage capabilities, each with individual requirements, e.g., with regards to processing latency or data locality. Third, the infrastructure for Edge Computing, i.e., the resources that will be used to host data and computations, are highly heterogeneous in their capacity, cost, and ownership.

This thesis consisted of four main parts that made contributions to the understanding, planning, deployment, and operation of Edge Computing. Our focus on *urban* Edge Computing is especially emphasized by Part II, in which we consider urban infrastructures, such as street lamps, as infrastructural resources for Edge Computing deployments. The contributions presented in the other parts of this thesis are relevant beyond urban infrastructures.

We now conclude this thesis by summarizing the contributions made in those parts.

Part I | The first part provided a detailed background and analysis of the state of the art in Edge Computing. We provided a taxonomy (Chapter 2) of Edge Computing and analyzed its characteristics (Chapter 3). In Chapter 4, we performed a systematic survey of use cases for Edge Computing and proposed a classification scheme for Edge Computing applications. Part I therefore contributes to the general understanding of the field of Edge Computing, refines its definition in an urban context, and highlights important (future) use cases.

Part II | In the second part, we examined the infrastructures in an urban environment on which we can place cloudlets—small-scale proximate data centers—to offer Edge Computing resources to (mobile) users. In Chapter 5, we performed a systematic analysis of the coverage that can be achieved using the existing access point infrastructure in a city, while Chapter 6 proposed a placement strategy for heterogeneous cloudlets on those access points. This part hence provided insights for the design and planning of the physical infrastructure (in terms of hardware and communication capabilities) for Edge Computing in an urban environment.

Part III | Part III focused on the actual *execution* of computations at the edge. To realize this, Chapter 7 proposed an Edge Computing framework that is based on *computation onloading* through a *microservice store*. The framework is an alternative approach to state-of-the-art offloading approaches, in which the offloadable parts of the applications are either pre-provisioned on the edge nodes or transferred from the client device to the surrogate. Furthermore, the reuse of microservice instances at runtime allows for an efficient use of edge resources. We have shown how our approach is beneficial in terms of reduced end-to-end latency and battery savings for the client device. The contributions in this part lay the basis for the efficient execution of application parts at the edge.

Part IV | Lastly, Part IV presented strategies and adaptations to make runtime decisions in an Urban Edge Computing system. In Chapter 8, we proposed an approach for the scalable placement of operators, i.e., functional parts of applications. Given the complexity of the problem, we suggested placement heuristics that reduce the solving time of the problem, while introducing only a small optimality gap. The reduction in solving time in practice leads to a faster provisioning of services for users, increasing the quality of experience

> and allowing for fast reconfigurations of placements. Chapter 9 in turn investigated the placement of *data*. More specifically, we proposed the concept of *context-aware micro storage* at the edge of the network. We realized this concept and developed *vStore*, an open-source framework targeted at mobile users. Our concept makes rule-based storage decisions, leverages proximate storage nodes to save core network bandwidth, and allows for cross-application sharing of data. Chapter 10 revised the previously introduced concept of microservices by making them adaptable at runtime. We introduced the notion of *service variants* that can dynamically be selected to trade off execution time and quality of results. Given the different latency requirements of applications and varying computing resources in Edge Computing, this tradeoff allows to adapt an Edge Computing execution system to those characteristics. Furthermore, the concept of service variants allows to shrink the gap in resource elasticity between Edge Computing and Cloud Computing.

The contributions made in the individual parts can stand for themselves and should be viewed independently. However, in a broader context, parts II–IV can be considered building blocks for an overall Urban Edge Computing system. Part II provides the substrate on which a distributed Edge Computing runtime (Part III) operates. The runtime in turn relies on mechanisms presented in Part IV for its decision-making (e.g., where to place data and carry out computations).

11.2 Future Work

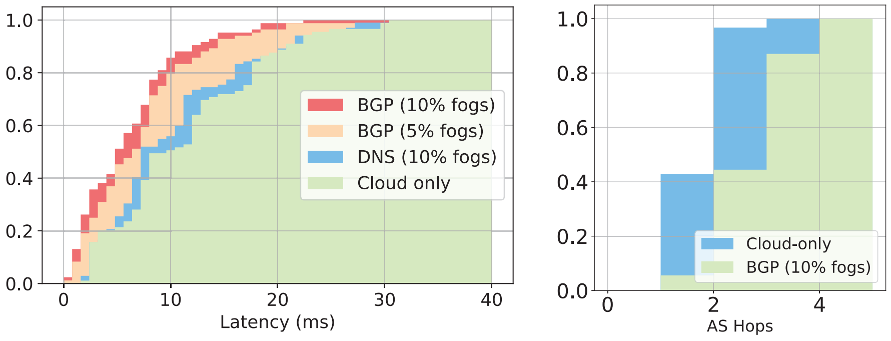
This thesis made distinctive improvements in the field of Edge Computing. We can, however, imagine several remaining obstacles that hinder the widespread availability of Edge Computing in urban spaces as envisioned in this thesis. In this chapter, we outline some open questions and possible future research directions.

11.2.1 Discovery

On several occasions, we have highlighted the opportunistic and highly distributed nature of Urban Edge Computing, in which users leverage various storage and computing resources owned by different stakeholders in their surroundings. One manifestation of this aspect is that surrogates will be spread throughout the entire network, inside different Internet autonomous systems (AS) [HB96] that are owned and operated by different stakeholders. This in itself poses challenges, e.g, with regards to interoperability and revenue models (see Section 11.2.3). Even if we assume a future standardized Edge Computing infrastructure with roaming and accounting models across stakeholders, the challenge of *discovering* available surrogates remains.

Service discovery is a well-established problem in the domains of pervasive computing and web services and, hence, various solutions have been proposed [Hel02; ZMN05]. For different reasons they are not applicable in an Edge Computing environment, where we require a loose coupling between users and surrogates in order to achieve a seamless migration of services (e.g., because of user mobility). In Edge Computing, we need discovery mechanisms that work on a global scale and yet remain scalable. Broadcast protocols and approaches that rely on centralized databases clearly do not fulfill these requirements. Several proposed approaches in the context of Edge Computing or IoT also rely on centralized approaches for the discovery [Bha+16; Che+18b] or assume that all resources for computing are known a priori [Xio+18; Cap+18]. Approaches that create overlays, e.g., through DHTs[1] [Ged+17; TVM18], suffer from topology mismatch, i.e., the overlay topology does not accurately represent the properties of the underlying network. Consequently, those approaches cannot make guarantees on the resulting latency—a crucial factor for many Edge Computing applications.

In conclusion, the scalable discovery of federated resources remains challenging. In a recent contribution [Ged+20], we proposed to jointly use on-path and off-path for the discovery in federated Fog Computing environments. We argue that the techniques presented there can be applied to an Urban Edge Computing environment, where we also have highly distributed resources in different administrative domains. The discovery mechanism combined DNS-based discovery (using the NAPTR resource records) with the announcement of computing resources through custom BGP community strings. Figure 11.1 shows the results with varying percentages of fog sites. The latency reduction can be seen from Figure 11.1(a) (comparing the latency of using only cloud resources with using fog resources as determined by the different proposed discovery methods). From our measurements, we could also observe that the latency does not correlate with the hop count (see Figure 11.1(b)). It remains to be investigated if this also applies to Edge Computing, where resources are often (co-)located at the wireless gateway. Furthermore, future work should investigate the efficiency with regards to the discovery time and the caching and dissemination of discovery results.



(a) CDF of the Latency (b) CDF of the AS Hop Count

FIGURE 11.1: RESULTS OF COMBINED DISCOVERY METHODS (FIGURE TAKEN FROM [GED+20])

11.2.2 Security, Privacy, and Trust

Distributing data and computations over a network naturally incurs challenges with respect to security, privacy, and trust. Some of these challenges are similar to those in Cloud Computing, while others specifically arise because of the characteristics of Edge Computing.

[1] Distributed Hash Tables

Mainly due to the high (geographic) distribution of resources, Edge Computing increases the likelihood of physical attacks on the infrastructure. Not only a few data centers, but a multitude of edge sites need to be secured. These edge sites are sometimes located in unguarded places, such as roadside units or street lamps, making them susceptible to theft and sabotage. Hence, ensuring the physical integrity of edge sites remains challenging.

Aside from the *physical* security of edge sites, securing the execution environment is another crucial factor in both Edge Computing and Cloud Computing. Edge Computing typically makes use of lightweight virtualization technologies (see Section 3.5.2). The security aspect of those different virtualization technologies is subject to discussion within the research community [Man+17; CMD16]. Other challenges include securing the users' data and network security [YQL15]. As an example for the latter, Stojmenovic et al. [Sto+16] describe a stealthy man-in-the-middle attack in a Fog Computing scenario. Roman et al. [RLM18] survey further security threats and challenges. Following the aftermath of a successful attack, Wang et al. [WUS15] discuss future challenges in forensics for Fog Computing.

Users of Edge Computing deployments have to trust the execution environment. In contrast to Cloud Computing, this execution environment might span over multiple administrative domains, including devices that are privately owned (see Section 3.3). Trust can encompass multiple aspects. First, edge nodes should not reveal privacy-sensitive information. Second, if we envision the deployment of an Edge Computing system as described in Chapter 7, we need to guarantee the security and integrity of the services provided, especially if those are shared between multiple applications and the user has no control over which service will be invoked. Furthermore, users should be able to trust the correctness of the computation. Few initial works in this direction exist. For example, Ruan et al. [RDU18] propose a trust management framework to assess trust for computations and devices in an MEC-IoT environment. Zhang et al. [Zha+18c] analyze the data security and privacy threats in Edge Computing. Others have focused on these issues solely in the context of the IoT [Vas+15; Per+15].

11.2.3 Business Models

Unlike for current Cloud Computing offerings, in Edge Computing it remains unclear what the predominant business models will be. In their review about the past research on cyber foraging, Balan and Flin [BF17] raise the question of who should provide surrogates for offloading, naming application providers, users, and third-party infrastructure providers as possible candidates. Aijaz et al. [AAA13] shed light on the business perspective of data offloading from the point of view of device manufacturers, service providers, and hotspot operators. Others have argued that Edge Computing needs the same flexible pay-as-you-go services models as Cloud Computing [Ahm+17].

Edge Computing as we have envisioned it throughout this thesis is dynamic and must be carried out in cooperative ways, e.g., mobile users must switch to other cloudlets. Hence, to realize the full potential of Edge Computing, different stakeholders and competitors must work together (e.g., to regulate data transit and enabling handover of users and services). Ideally, users should be able to seamlessly migrate their data and computations independent of who owns the underlying infrastructure. Besides new business models and technical standards, this also requires novel accounting mechanisms [Rez+18]. We believe this is one of the main reasons that no widespread infrastructure for Urban Edge Computing is available today.

If we also consider privately owned devices for Edge Computing, the question of business models shifts to incentive mechanisms. Previous works in this aspect have focused on sharing the access network, e.g., sharing one's broadband connection via WiFi [Shi+15; MPP10]. Sharing the additional resources available at or colocated with home gateways could be a next step. Some recent works have begun to investigate incentive mechanisms for offloading in Edge Computing [Zen+18; Liu+17] but those are mainly theoretical results that still are to be validated with practical studies.

11.3 Outlook

Although Edge Computing is not ubiquitously and openly present today, novel applications such as mobile augmented reality and assisted driving will drive the need for low-latency processing over fast wireless connections in the near future, especially when those applications work cooperatively.

We are just beginning to see how infrastructure is changing to support such applications. Novel communication standards such as 5G are being deployed, offering unprecedented wireless bandwidth and ultra-low latency connectivity for mobile devices. At the same time, the computing resources at the edge change and now include more general-purpose, virtualized computing hardware, resembling the infrastructure found in the Cloud Computing environment. Owners of these resources (e.g., mobile network operators) therefore can leverage this flexibility to offer additional services, e.g., general-purpose computing for edge applications. Established Cloud Computing providers are also offering new services for proximate computing resources (e.g., Amazon's *lambda@edge*[2] or Microsoft's announcement of *Azure Edge Zones*[3]) in an attempt to enter the emerging Edge Computing market. Both of these trends—enhanced wireless access technologies and more proximate general-purpose computing resources— lay important technological groundworks for the practical adoption of Edge Computing.

In the 2019 edition[4] of their "Hype Cycle for Emerging Technologies", Gartner assessed *edge analytics* at the "peak of inflated expectations". How deep the drop from this peak will be before Edge Computing reaches the "plateau of productivity" will depend on a number of factors. Some of those remain open questions, especially w.r.t. business models (see Section 11.2.3) and compatibility between different Edge Computing providers, while this dissertation contributed solutions w.r.t. technical questions for Edge Computing.

[2] https://aws.amazon.com/lambda/edge/ (accessed: 2020-06-12)

[3] https://docs.microsoft.com/en-us/azure/networking/edge-zones-overview (accessed: 2020-06-12)

[4] https://www.gartner.com/en/documents/3956015/hype-cycle-for-emerging-technologies-2019 (accessed: 2020-06-11)

Bibliography

[AAA13] Adnan Aijaz, Hamid Aghvami, and Mojdeh Amani. "A survey on mobile data offloading: technical and business perspectives". In: *IEEE Wireless Communnications* 20.2 (2013), pp. 104–112.

[Aaz+16] M. Aazam, M. St-Hilaire, C. Lung, and I. Lambadaris. "Cloud-based smart waste management for smart cities". In: *Proc. of the 2016 IEEE 21st International Workshop on Computer Aided Modelling and Design of Communication Links and Networks (CAMAD)*. 2016, pp. 188–193.

[Aba+] Omid Abari, Dinesh Bharadia, Austin Duffield, and Dina Katabi. "Enabling High-Quality Untethered Virtual Reality". In: *Proc. of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 531–544.

[Aba+15] Omid Abari, Deepak Vasisht, Dina Katabi, and Anantha Chandrakasan. "Caraoke: An E-Toll Transponder Network for Smart Cities". In: *Proc. of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 2015, pp. 297–310.

[Abb+18] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. "Mobile Edge Computing: A Survey". In: *IEEE Internet of Things Journal* 5.1 (2018), pp. 450–465.

[Abd+16] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati. "Network function virtualization in 5G". In: *IEEE Communications Magazine* 54.4 (2016), pp. 84–91.

[Abo+99] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. "Towards a Better Understanding of Context and Context-Awareness". In: *Proc. of the First International Symposium on Handheld and Ubiquitous Computing (HUC)*. 1999, pp. 304–307.

[AD14] H. Ahlehagh and S. Dey. "Video-Aware Scheduling and Caching in the Radio Access Network". In: *IEEE/ACM Transactions on Networking* 22.5 (2014), pp. 1444–1462.

[ADH18] Nurzaman Ahmed, Debashis De, and Iftekhar Hussain. “Internet of Things (IoT) for Smart Precision Agriculture and Farming in Rural Areas”. In: *IEEE Internet of Things Journal* 5.6 (2018), pp. 4890–4899.

[AG10] Alvaro Alesanco and Jose García. “Clinical Assessment of Wireless ECG Transmission in Real-Time Cardiac Telemonitoring”. In: *IEEE Transactions on Information Technology in Biomedicine* 14 (2010), pp. 1144–1152.

[Agr+16] Ankur Agrawal, Jungwook Choi, Kailash Gopalakrishnan, Suyog Gupta, Ravi Nair, Jinwook Oh, Daniel A. Prener, Sunil Shukla, Vijayalakshmi Srinivasan, and Zehra Sura. “Approximate computing: Challenges and opportunities”. In: *Proc. of the IEEE International Conference on Rebooting Computing (ICRC)*. 2016, pp. 1–8.

[Agu+10] Trevor R. Agus, Clara Suied, Simon J. Thorpe, and Daniel Pressnitzer. “Characteristics of human voice processing”. In: *Proc. of the International Symposium on Circuits and Systems (ISCAS)*. 2010, pp. 509–512.

[AH14] M. Aazam and E. Huh. “Fog Computing and Smart Gateway Based Communication for Cloud of Things”. In: *Proc. of the 2014 International Conference on Future Internet of Things and Cloud*. 2014, pp. 464–470.

[AH15] M. Aazam and E. Huh. “E-HAMC: Leveraging Fog computing for emergency alert service”. In: *Proc. of the 2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. 2015, pp. 518–523.

[Ahm+17] E. Ahmed, A. Ahmed, I. Yaqoob, J. Shuja, A. Gani, M. Imran, and M. Shoaib. “Bringing Computation Closer toward the User Network: Is Edge Computing the Solution?” In: *IEEE Communications Magazine* 55.11 (2017), pp. 138–144.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. “The Internet of Things: A survey”. In: *Computer Networks* 54.15 (2010), pp. 2787–2805.

[AKL18] Haider A. F. Almurib, Thulasiraman Nandha Kumar, and Fabrizio Lombardi. “Approximate DCT Image Compression Using Inexact Computing”. In: *IEEE Transactions on Computers* 67.2 (2018), pp. 149–159.

[Ala+10] Firat Alagöz, André Calero Valdez, Wiktoria Wilkowska, Martina Ziefle, Stefan Dorner, and Andreas Holzinger. “From Cloud Computing to Mobile Internet, from User Focus to Culture and Hedonism - The Crucible of Mobile Health Care and Wellness Applications”. In: *Proc. of the 5th International Conference on Pervasive Computing and Applications (ICPCA)*. 2010, pp. 38–45.

[Ala+18] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen. “Orchestration of Microservices for IoT Using Docker and Edge Computing”. In: *IEEE Communications Magazine* 56.9 (2018), pp. 118–123.

[AlF+15] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications". In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2347–2376.

[Alr+17] A. Alrawais, A. Alhothaily, C. Hu, and X. Cheng. "Fog Computing for the Internet of Things: Security and Privacy Issues". In: *IEEE Internet Computing* 21.2 (2017), pp. 34–42.

[Alt+16] Qutaibah Althebyan, Qussai Yaseen, Yaser Jararweh, and Mahmoud Al-Ayyoub. "Cloud support for large scale e-healthcare systems". In: *Annales des Télécommunications* 71.9-10 (2016), pp. 503–515.

[AMM15] Sereen Althaher, Pierluigi Mancarella, and Joseph Mutale. "Automated demand response from home energy management system under dynamic pricing and power and comfort constraints". In: *IEEE Transactions on Smart Grid* 6.4 (2015), pp. 1874–1883.

[Ana+17] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha. "Real-Time Video Analytics: The Killer App for Edge Computing". In: *Computer* 50.10 (2017), pp. 58–67.

[App+10] David Applegate, Aaron Archer, Vijay Gopalakrishnan, Seungjoon Lee, and K. K. Ramakrishnan. "Optimal Content Placement for a Large-scale VoD System". In: *Proc. of the 2010 ACM Conference on Emerging Networking Experiments (CoNEXT)*. 2010, 4:1–4:12.

[AS13] Sameer Alawnah and Assim Sagahyroon. "Modeling Smartphones Power". In: *Proc. of Eurocon*. 2013, pp. 369–374.

[AS16] Aymen El Amraoui and Kaouthar Sethom. "Cloudlet Softwarization for Pervasive Healthcare". In: *Proc. of the 30th International Conference on Advanced Information Networking and Applications (AINA) Workshops*. 2016, pp. 628–632.

[ASH15] Nazanin Aminzadeh, Zohreh Sanaei, and Siti Hafizah Ab Hamid. "Mobile storage augmentation in mobile cloud computing: Taxonomy, approaches, and open issues". In: *Simulation Modelling Practice and Theory* 50 (2015), pp. 96–108.

[Awa+19] K. S. Awaisi, A. Abbas, M. Zareei, H. A. Khattak, M. U. Shahid Khan, M. Ali, I. Ud Din, and S. Shah. "Towards a Fog Enabled Efficient Car Parking Architecture". In: *IEEE Access* 7 (2019), pp. 159100–159111.

[AXS17] Raef Abdallah, Lanyu Xu, and Weisong Shi. "Lessons and experiences of a DIY smart home". In: *Proc. of the Workshop on Smart Internet of Things*. SmartIoT '17. 2017, 4:1–4:6.

[AZH18] Mohammad Aazam, Sherali Zeadally, and Khaled A Harras. "Deploying fog computing in industrial internet of things and industry 4.0". In: *Transactions on Industrial Informatics* 14.10 (2018), pp. 4674–4682.

[Azi+17] Iman Azimi, Arman Anzanpour, Amir M. Rahmani, Tapio Pahikkala, Marco Levorato, Pasi Liljeberg, and Nikil D. Dutt. "HiCH: Hierarchical Fog-Assisted Computing Architecture for Healthcare IoT". In: *ACM Transactions on Embedded Computing Systems* 16.5 (2017), 174:1–174:20.

[AZM15] T. Anagnostopoulos, A. Zaslavsky, and A. Medvedev. “Robust waste collection exploiting cost efficiency of IoT potentiality in Smart Cities”. In: *Proc. of the 2015 International Conference on Recent Advances in Internet of Things (RIoT)*. 2015, pp. 1–6.

[Azu97] Ronald Azuma. “A Survey of Augmented Reality”. In: *Presence* 6.4 (1997), pp. 355–385.

[Baj+15] A. Bajpai, V. Jilla, V. N. Tiwari, S. M. Venkatesan, and R. Narayanan. “Quantifiable fitness tracking using wearable devices”. In: *Proc. of the 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. 2015, pp. 1633–1637.

[Bal+02] Rajesh Krishna Balan, Jason Flinn, Mahadev Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. “The case for cyber foraging”. In: *Proc. of the 10th ACM SIGOPS European Workshop*. 2002, pp. 87–92.

[Bal+07] Rajesh Krishna Balan, Darren Gergle, Mahadev Satyanarayanan, and James Herbsleb. “Simplifying Cyber Foraging for Mobile Devices”. In: *Proc. of the 5th International Conference on Mobile Systems, Applications and Services*. MobiSys ’07. 2007, pp. 272–285.

[Bal+17] Ioana Baldini, Paul C. Castro, Kerry Shih-Ping Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. “Serverless Computing: Current Trends and Open Problems”. In: *Research Advances in Cloud Computing*. 2017, pp. 1–20.

[Ban+14] B. Bangerter, S. Talwar, R. Arefi, and K. Stewart. “Networks and devices for the 5G era”. In: *IEEE Communications Magazine* 52.2 (2014), pp. 90–96.

[Bas+03] Harpal S. Bassali, Krishnanand M. Kamath, Rajendraprasad B. Hosamani, and Lixin Gao. “Hierarchy-aware algorithms for CDN proxy placement in the Internet”. In: *Computer Communications* 26.3 (2003), pp. 251–263.

[Baz+13] Sobir Bazarbayev, Matti A. Hiltunen, Kaustubh R. Joshi, William H. Sanders, and Richard D. Schlichting. “PSCloud: a durable context-aware personal storage cloud”. In: *Proc. of the 9th Workshop on Hot Topics in Dependable Systems (HotDep)*. 2013, 9:1–9:6.

[BB14] M. Bani Younes and A. Boukerche. “An Intelligent Traffic Light scheduling algorithm through VANETs”. In: *Proc. of the 39th Annual IEEE Conference on Local Computer Networks Workshops*. 2014, pp. 637–642.

[BBD14] E. Bastug, M. Bennis, and M. Debbah. “Living on the edge: The role of proactive caching in 5G wireless networks”. In: *IEEE Communications Magazine* 52.8 (2014), pp. 82–89.

[BCZ97] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura. “An architecture for active networking”. In: *Proc. of the 7th IFIP International Conference on High Performance Networking (HPN’97)*. 1997, pp. 265–279.

[BD07] Danilo Beuche and Mark Dalgarno. “Software product line engineering with feature models”. In: *Overload Journal* 78 (2007), pp. 5–8.

[BD17] Arani Bhattacharya and Pradipta De. “A survey of adaptation techniques in computation offloading”. In: *Journal of Network and Computer Applications* 78 (2017), pp. 97–115.

[Bec+14] Michael Till Beck, Martin Werner, Sebastian Feld, and Thomas Schimper. “Mobile Edge Computing: A Taxonomy”. In: *Proc. of the Sixth International Conference on Advances in Future Internet (AFIN)*. 2014, pp. 48–54.

[Ben14] Juan Benet. “IPFS - Content Addressed, Versioned, P2P File System”. In: *CoRR* abs/1407.3561 (2014), pp. 1–11.

[BF17] Rajesh Krishna Balan and Jason Flinn. “Cyber Foraging: Fifteen Years Later”. In: *IEEE Pervasive Computing* 16.3 (2017), pp. 24–30.

[BG17] Tayebeh Bahreini and Daniel Grosu. “Efficient placement of multicomponent applications in edge computing systems”. In: *Proc. of the 2nd ACM/IEEE Symposium on Edge Computing (SEC)*. 2017, 5:1–5:11.

[BGG19] Azzedine Boukerche, Shichao Guan, and Robson E. De Grande. “Sustainable Offloading in Mobile Cloud Computing: Algorithmic Design and Implementation”. In: *ACM Computing Surveys* 52.1 (2019), 11:1–11:37.

[BGT16] B. Butzin, F. Golatowski, and D. Timmermann. “Microservices approach for the internet of things”. In: *Proc. of the 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2016, pp. 1–6.

[Bha+15a] Ketan Bhardwaj, Pragya Agarwal, Ada Gavrilovska, Karsten Schwan, and Adam Allred. “AppFlux: Taming App Delivery via Streaming”. In: *Proc. of USENIX Conference on Timely Results in Operating Systems (TRIOS)*. Oct. 2015, pp. 1–14.

[Bha+15b] Ketan Bhardwaj, Pragya Agrawal, Ada Gavrilovska, and Karsten Schwan. “AppSachet: Distributed App Delivery from the Edge Cloud”. In: *Proc. of the 7th International Conference on Mobile Computing, Applications and Services (MobiCASE)*. 2015, pp. 89–106.

[Bha+16] K. Bhardwaj, M. Shih, P. Agarwal, A. Gavrilovska, T. Kim, and K. Schwan. “Fast, Scalable and Secure Onloading of Edge Functions Using AirBox”. In: *Proc. of the 2016 IEEE/ACM Symposium on Edge Computing (SEC)*. 2016, pp. 14–27.

[BHJ16] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture”. In: *IEEE Software* 33.3 (2016), pp. 42–52.

[Bi+18] Yuanguo Bi, Guangjie Han, Chuan Lin, Qingxu Deng, Lei Guo, and Fuliang Li. “Mobility Support for Fog Computing: An SDN Approach”. In: *IEEE Communications Magazine* 56.5 (2018), pp. 53–59.

[BI14] Benjamin Billet and Valérie Issarny. “From Task Graphs to Concrete Actions: A New Task Mapping Algorithm for the Future Internet of Things”. In: *Proc. of the 11th IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. 2014, pp. 470–478.

[BKM09] Aaron Bangor, Philip Kortum, and James Miller. “Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale”. In: *Journal of Usability Studies* 4.3 (2009), pp. 114–123.

[BLS17] Sultan Basudan, Xiaodong Lin, and Karthik Sankaranarayanan. “A Privacy-Preserving Vehicular Crowdsensing-Based Road Surface Condition Monitoring System Using Fog Computing”. In: *IEEE Internet of Things Journal* 4.3 (2017), pp. 772–782.

[BMM14] James Bornholt, Todd Mytkowicz, and Kathryn S. McKinley. “Uncertain <T>: A First-order Type for Uncertain Data”. In: *Proc. of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Ed. by Rajeev Balasubramonian, Al Davis, and Sarita V. Adve. 2014, pp. 51–66.

[BMV10] Aruna Balasubramanian, Ratul Mahajan, and Arun Venkataramani. “Augmenting Mobile 3G Using WiFi”. In: *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’10. 2010, pp. 209–222.

[BOE17] Ahmet Cihat Baktir, Atay Ozgovde, and Cem Ersoy. “How Can Edge Computing Benefit From Software-Defined Networking: A Survey, Use Cases, and Future Directions”. In: *IEEE Communications Surveys and Tutorials* 19.4 (2017), pp. 2359–2391.

[Boh+15] Steven Bohez, Jaron Couvreur, Bart Dhoedt, and Pieter Simoens. “Cloudlet-based Large-scale 3D Reconstruction Using Real-time Data from Mobile Depth Cameras”. In: *Proc. of the 6th International Workshop on Mobile Cloud Computing (MCS)*. 2015, pp. 8–14.

[Bon+12] Flavio Bonomi, Rodolfo A. Milito, Jiang Zhu, and Sateesh Addepalli. “Fog computing and its role in the internet of things”. In: *Proc. of the 1st Workshop on Mobile Cloud Computing (MCC)*. 2012, pp. 13–16.

[Bra+17a] Tristan Braud, Farshid Hassani Bijarbooneh, Dimitris Chatzopoulos, and Pan Hui. “Future Networking Challenges: The Case of Mobile Augmented Reality”. In: *Proc. of the 37th IEEE International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 1796–1807.

[Bra+17b] P. J. Braun, S. Pandi, R. Schmoll, and F. H. P. Fitzek. “On the study and deployment of mobile edge cloud for tactile Internet using a 5G gaming application”. In: *Proc. of the 2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 2017, pp. 154-1-59.

[Bre+19] Martin Breitbach, Dominik Schäfer, Janick Edinger, and Christian Becker. “Context-Aware Data and Task Placement in Edge Computing Environments”. In: *Proc. of the 2019 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 2019, pp. 1–10.

[Bro96] John Brooke. “SUS-A quick and dirty usability scale”. In: *Usability evaluation in industry* 189.194 (1996), pp. 4–7.

[BS13] Eyuphan Bulut and Boleslaw K. Szymanski. “WiFi access point deployment for efficient mobile data offloading”. In: *Mobile Computing and Communications Review* 17.1 (2013), pp. 71–78.

[BS16] E. Bulut and B. Szymanski. “Rethinking offloading WiFi access point deployment from user perspective”. In: *Proc. of the 12th IEEE International Conference on Wireless and Mobile Computing (WiMob)*. 2016, pp. 1–6.

[BSC10] David Benavides, Sergio Segura, and Antonio Ruiz Cortés. “Automated analysis of feature models 20 years later: A literature review”. In: *Information Systems* 35.6 (2010), pp. 615–636.

[Bur+15] V. Burger, M. Seufert, F. Kaup, M. Wichtlhuber, D. Hausheer, and P. Tran-Gia. “Impact of WiFi offloading on video streaming QoE in urban environments”. In: *Proc- of the International Conference on Communication Workshops (ICC Workshops)*. 2015, pp. 1717–1722.

[BZ11] Nicola Bui and Michele Zorzi. “Health care applications: a solution based on the internet of things”. In: *Proc. of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies (ISABEL)*. 2011, 131:1–131:5.

[BZ17] Paolo Bellavista and Alessandro Zanni. “Feasibility of Fog Computing Deployment based on Docker Containerization over RaspberryPi”. In: *Proc. of the 18th International Conference on Distributed Computing and Networking (ICDCN)*. 2017, pp. 1–10.

[CAG08] V. Chandrasekhar, J. G. Andrews, and A. Gatherer. “Femtocell networks: a survey”. In: *IEEE Communications Magazine* 46.9 (2008), pp. 59–67.

[Cai+18] W. Cai, F. Chi, X. Wang, and V. C. M. Leung. “Toward Multiplayer Cooperative Cloud Gaming”. In: *IEEE Cloud Computing* 5.5 (2018), pp. 70–80.

[Cao+15] Yu Cao, Songqing Chen, Peng Hou, and Donald Brown. “FAST: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation”. In: *Proc. of the 10th IEEE International Conference on Networking, Architecture and Storage (NAS)*. 2015, pp. 2–11.

[Cap+18] Justin Cappos, Matthew Hemmings, Rick McGeer, Albert Rafetseder, and Glenn Ricart. “EdgeNet: A Global Cloud That Spreads by Local Action”. In: *Proc. of the IEEE/ACM Symposium on Edge Computing (SEC)*. 2018, pp. 359–360.

[Car+12] Ben W. Carabelli, Andreas Benzing, Frank Dürr, Boris Koldehofe, Kurt Rothermel, Georg S. Seyboth, Rainer Blind, Mathias Bürger, and Frank Allgöwer. “Exact convex formulations of network-oriented optimal operator placement”. In: *Proc. of the 51th IEEE Conference on Decision and Control (CDC)*. 2012, pp. 3777–3782.

[Car+15] V. Cardellini, V. Grassi, F. L. Presti, and M. Nardelli. “On QoS-aware scheduling of data stream applications over fog computing infrastructures”. In: *Proc. of the 2015 IEEE Symposium on Computers and Communication (ISCC)*. 2015, pp. 271–276.

[Car+16] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. “Optimal operator placement for distributed stream processing applications”. In: *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems (DEBS)*. 2016, pp. 69–80.

[Car+17] Marcel Caria, Jasmin Schudrowitz, Admela Jukan, and Nicole Kemper. “Smart farm computing systems for animal welfare monitoring”. In: *Proc. of the 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2017, pp. 152–157.

[Cas14] Marco Casini. “Internet of things for Energy efficiency of buildings”. In: *International Scientific Journal Architecture and Engineering* 2.1 (2014), pp. 24–28.

[CCK18] Eduardo Cuervo, Krishna Chintalapudi, and Manikanta Kotaru. “Creating the perfect Illusion: What will it take to Create Life-Like Virtual Reality Headsets?” In: *Proc. of the 19th International Workshop on Mobile Computing Systems & Applications (HotMobile)*. 2018, pp. 7–12.

[CCL14] W. Cai, M. Chen, and V. C. M. Leung. “Toward Gaming as a Service”. In: *IEEE Internet Computing* 18.3 (2014), pp. 12–18.

[CCP19] Claudio Cicconetti, Marco Conti, and Andrea Passarella. “Low-latency Distributed Computation Offloading for Pervasive Environments”. In: *Proc. of the 2019 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 2019, pp. 1–10.

[CDO17] Vittorio Cozzolino, Aaron Yi Ding, and Jörg Ott. “FADES: Fine-Grained Edge Offloading with Unikernels”. In: *Proc. of the Workshop on Hot Topics in Container Networking and Networked Systems (HotConNet)*. 2017, pp. 36–41.

[CDO19] Vittorio Cozzolino, Aaron Yi Ding, and Jörg Ott. “Edge Chaining Framework for Black Ice Road Fingerprinting”. In: *Proc. of the 2nd International Workshop on Edge Systems, Analytics and Networking (EdgeSys)*. 2019, pp. 42–47.

[CFB14] Francesco Calabrese, Laura Ferrari, and Vincent D. Blondel. “Urban Sensing Using Mobile Phone Network Data: A Survey of Research”. In: *ACM Compututing Surveys* 47.2 (2014), 25:1–25:20.

[Cha+14] Hyunseok Chang, Adiseshu Hari, Sarit Mukherjee, and T. V. Lakshman. “Bringing the cloud to the edge”. In: *Proc. of the IEEE INFOCOM Workshops*. 2014, pp. 346–351.

[Cha+15] Amir Chaudhry, Jon Crowcroft, Heidi Howard, Anil Madhavapeddy, Richard Mortier, Hamed Haddadi, and Derek McAuley. “Personal Data: Thinking Inside the Box”. In: *Proc. of the 5th Decennial Aarhus Conference on Critical Alternatives*. CA ’15. 2015, pp. 29–32.

[Cha+16] David Chaum, Farid Javani, Aniket Kate, Anna Krasnova, Joeri de Ruiter, Alan T Sherman, and D Das. “cMix: Anonymization by high-performance scalable mixing”. In: *Proc. of USENIX Security*. 2016, pp. 1–19.

[Che+03] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Cetintemel, Ying Xing, and Stanley B. Zdonik. “Scalable Distributed Stream Processing”. In: *Proc. of the First Biennial Conference on Innovative Data Systems Research (CIDR)*. 2003, pp. 1–12.

[Che+16] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. “Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing”. In: *IEEE/ACM Transactions on Networking* 24.5 (2016), pp. 2795–2808.

[Che+17a] Ning Chen, Yu Chen, Xinyue Ye, Haibin Ling, Sejun Song, and Chin-Tser Huang. “Smart City Surveillance in Fog Computing”. In: *Advances in Mobile Cloud Computing and Big Data in the 5G Era*. Ed. by Constandinos X. Mavromoustakis, George Mastorakis, and Ciprian Dobre. Springer International Publishing, 2017, pp. 203–226.

[Che+17b] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta L. Klatzky, Daniel P. Siewiorek, and Mahadev Satyanarayanan. “An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance”. In: *Proc. of the Second ACM/IEEE Symposium on Edge Computing (SEC)*. 2017, 14:1–14:14.

[Che+18a] Chia-Yu Chen, Jungwook Choi, Kailash Gopalakrishnan, Viji Srinivasan, and Swagath Venkataramani. “Exploiting approximate computing for deep learning acceleration”. In: *Proc. of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2018, pp. 821–826.

[Che+18b] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa. “FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities”. In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 696–707.

[Chi+13] Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. “Analysis and characterization of inherent application resilience for approximate computing”. In: *Proc. of the 50th Annual Design Automation Conference (DAC)*. 2013, 113:1–113:9.

[Cho+12] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. “The Brewing Storm in Cloud Gaming: A Measurement Study on Cloud to End-user Latency”. In: *Proc. of the 11th Annual Workshop on Network and Systems Support for Games*. NetGames ’12. 2012, 2:1–2:6.

[Cho+16] Junguk Cho, Karthikeyan Sundaresan, Rajesh Mahindra, Jacobus Van der Merwe, and Sampath Rangarajan. “ACACIA: Context-aware Edge Computing for Continuous Interactive Applications over Mobile Networks”. In: *Proc. of the 12th International on Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. 2016, pp. 375–389.

[Chu+11] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. “CloneCloud: elastic execution between mobile device and cloud”. In: *Proc. of the 6th European Conference on Computer Systems (EuroSys)*. 2011, pp. 301–314.

[Chu+13] Chit Chung, Dennis Egan, Ashish Jain, Nicholas Caruso, Colin Misner, and Richard Wallace. “A Cloud-Based Mobile Computing Applications Platform for First Responders”. In: *Proc. of the 7th IEEE International Symposium on Service-Oriented System (SOSE)*. 2013, pp. 503–508.

[CLK07] S. H. Chang, H. J. La, and S. D. Kim. “A Comprehensive Approach to Service Adaptation”. In: *Proc. of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA ’07)*. 2007, pp. 191–198.

[CLP17] 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: *Proc. of the 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. 2017, pp. 41–50.

[CM12] Gianpaolo Cugola and Alessandro Margara. “Processing Flows of Information: From Data Stream to Complex Event Processing”. In: *ACM Computing Surveys* 44.3 (2012), 15:1–15:62.

[CM13] Gianpaolo Cugola and Alessandro Margara. “Deployment strategies for distributed complex event processing”. In: *Computing* 95.2 (2013), pp. 129–156.

[CMD16] T. Combe, A. Martin, and R. Di Pietro. “To Docker or Not to Docker: A Security Perspective”. In: *IEEE Cloud Computing* 3.5 (2016), pp. 54–62.

[Coh+15] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz. “Near optimal placement of virtual network functions”. In: *Proc. of the 2015 IEEE Conference on Computer Communications (INFOCOM)*. 2015, pp. 1346–1354.

[CP15] Zhen Cao and Panagiotis Papadimitriou. “Collaborative content caching in wireless edge with SDN”. In: *Proc. of the 1st Workshop on Content Caching and Delivery in Wireless Networks (CCDWN@CoNEXT)*. 2015, 6:1–6:7.

[CPS15] Alberto Ceselli, Marco Premoli, and Stefano Secci. “Cloudlet network design optimization”. In: *Proc. of the 14th IFIP Networking Conference*. 2015, pp. 1–9.

[CS15] Luca Canzian and Mihaela van der Schaar. “Real-time stream mining: online knowledge extraction using classifier networks”. In: *IEEE Network* 29.5 (2015), pp. 10–16.

[Cue+10] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. “MAUI: making smartphones last longer with code offload”. In: *Proc. of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2010, pp. 49–62.

[Cur+02] Francisco Curbera, Matthew J. Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. “Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI”. In: *IEEE Internet Computing* 6.2 (2002), pp. 86–93.

[CZZ13] Xuejun Cai, Shunliang Zhang, and Yunfei Zhang. “Economic analysis of cache location in mobile network”. In: *Proc. of the 2013 IEEE Wireless Communications and Networking Conference (WCNC)*. 2013, pp. 1243–1248.

[Dab+04] Frank Dabek, Russ Cox, M. Frans Kaashoek, and Robert Tappan Morris. “Vivaldi: A decentralized network coordinate system”. In: *Proc. of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2004, pp. 15–26.

[Dav+16] Nigel Davies, Nina Taft, Mahadev Satyanarayanan, Sarah Clinch, and Brandon Amos. “Privacy Mediators: Helping IoT Cross the Chasm”. In: *Proc. of the 17th International Workshop on Mobile Computing Systems and Applications*. HotMobile ’16. 2016, pp. 39–44.

[DBH15] S. K. Datta, C. Bonnet, and J. Haerri. “Fog Computing architecture to enable consumer centric Internet of Things services”. In: *Proc. of the 2015 International Symposium on Consumer Electronics (ISCE)*. 2015, pp. 1–2.

[Dey01] Anind K. Dey. “Understanding and Using Context”. In: *Personal and Ubiquitous Computing* 5.1 (2001), pp. 4–7.

[Dez+12] Niloofar Dezfuli, Jochen Huber, Simon Olberding, and Max Mühlhäuser. “CoStream: in-situ co-construction of shared experiences through mobile video sharing during live events”. In: *Proc. of the 2012 ACM annual conference on Human Factors in Computing Systems Extended Abstracts (CHI EA)*. 2012, pp. 2477–2482.

[DH15] Han Deng and I-Hong Hou. “Online scheduling for delayed mobile offloading”. In: *Proc. of the IEEE Conference on Computer Communications (INFOCOM)*. 2015, pp. 1867–1875.

[Dil+02] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. “Globally distributed content delivery”. In: *IEEE Internet Computing* 6.5 (2002), pp. 50–58.

[Dim+11] Savio Dimatteo, Pan Hui, Bo Han, and Victor O. K. Li. “Cellular Traffic Offloading through WiFi Networks”. In: *Proc. of the IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS)*. 2011, pp. 192–201.

[DIN18] DIN. *Integrated multi-functional Humble Lamppost (imHLa)*. Norm DIN SPEC 91347. 2018.

[Don+11] Yuan Dong, Haiyang Zhu, Jinzhan Peng, Fang Wang, Michael P. Mesnier, Dawei Wang, and Sun C. Chan. “RFS: a network file system for mobile devices and the cloud”. In: *Operating Systems Review* 45.1 (2011), pp. 101–111.

[Dra+17] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch-Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. “Microservices: Yesterday, Today, and Tomorrow”. In: *Present and Ulterior Software Engineering*. 2017, pp. 195–216.

[Dra+18] Nicola Dragoni, Ivan Lanese, Stephan Thordal Larsen, Manuel Mazzara, Ruslan Mustafin, and Larisa Safina. “Microservices: How To Make Your Application Scale”. In: *Proc. of the 11th International Andrei P. Ershov Informatics Conference (PSI)*. 2018, pp. 95–104.

[Dro+17] Utsav Drolia, Katherine Guo, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. "Cachier: Edge-Caching for Recognition Applications". In: *Proc. of the 37th IEEE International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 276–286.

[Dur+15] Francisco Rodrigo Duro, Francisco Javier García Blas, Daniel Higuero, Oscar Pérez, and Jesús Carretero. "CoSMiC: A hierarchical cloudlet-based storage architecture for mobile clouds". In: *Simulation Modelling Practice and Theory* 50 (2015), pp. 3–19.

[Dut+17] Joy Dutta, Chandreyee Chowdhury, Sarbani Roy, Asif Iqbal Middya, and Firoj Gazi. "Towards Smart City: Sensing Air Quality in City Based on Opportunistic Crowd-sensing". In: *Proc. of the 18th International Conference on Distributed Computing and Networking (ICDCN)*. ICDCN '17. 2017, 42:1–42:6.

[Ebe+16] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolás Serrano. "DevOps". In: *IEEE Software* 33.3 (2016), pp. 94–100.

[Edi+17] Janick Edinger, Dominik Schäfer, Martin Breitbach, and Christian Becker. "Developing distributed computing applications with Tasklets". In: *Proc. of the 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 2017, pp. 94–96.

[EFP10] Elias C. Efstathiou, Pantelis A. Frangoudis, and George C. Polyzos. "Controlled Wi-Fi Sharing in Cities: A Decentralized Approach Relying on Indirect Reciprocity". In: *IEEE Transactions on Mobile Computing* 9.8 (2010), pp. 1147–1160.

[EL16] R. Eidenbenz and T. Locher. "Task allocation for distributed stream processing". In: *Proc. of the 35th Annual IEEE International Conference on Computer Communications (INFOCOM)*. 2016, pp. 1–9.

[Elb+18] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler. "Toward Low-Latency and Ultra-Reliable Virtual Reality". In: *IEEE Network* 32.2 (2018), pp. 78–84.

[Eri13] Ericsson. "Ericsson Mobility Report". In: (2013), pp. 1–28. URL: `https://metis2020.com/res/docs/2013/ericsson-mobility-report-june-2013.pdf`.

[ERS16] Guy Even, Matthias Rost, and Stefan Schmid. "An Approximation Algorithm for Path Computation and Function Placement in SDNs". In: *Proc. of the 23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. 2016, pp. 374–390.

[Esp+17] F. Esposito, A. Cvetkovski, T. Dargahi, and J. Pan. "Complete edge function onloading for effective backend-driven cyber foraging". In: *Proc. of the 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2017, pp. 1–8.

[Eug+03] Patrick Th. Eugster, Pascal Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. "The many faces of publish/subscribe". In: *ACM Computing Surveys* 35.2 (2003), pp. 114–131.

[EZB17] Ahmed Elmokashfi, Dong Zhou, and Džiugas Baltrünas. “Adding the Next Nine: An Investigation of Mobile Broadband Networks Availability”. In: *Proc. of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom)*. 2017, pp. 88–100.

[FA17] Qiang Fan and Nirwan Ansari. “Cost Aware cloudlet Placement for big data processing at the edge”. In: *Proc. of the IEEE International Conference on Communications (ICC)*. 2017, pp. 1–6.

[Fas+07] Elena Fasolo, Michele Rossi, Jörg Widmer, and Michele Zorzi. “In-network aggregation techniques for wireless sensor networks: A survey”. In: *IEEE Wireless Communications* 14.2 (2007), pp. 70–87.

[Fel+15] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. “An updated performance comparison of virtual machines and Linux containers”. In: *Proc. of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2015, pp. 171–172.

[Fel+18] Patrick Felka, Artur Sterz, Katharina Keller, Bernd Freisleben, and Oliver Hinz. “The Context Matters: Predicting the Number of In-game Actions Using Traces of Mobile Augmented Reality Games”. In: *Proc. of the 17th International Conference on Mobile and Ubiquitous Multimedia*. MUM 2018. 2018, pp. 25–35.

[Fer+17] H. Fereidooni, T. Frassetto, M. Miettinen, A. Sadeghi, and M. Conti. “Fitness Trackers: Fit for Health but Unfit for Security and Privacy”. In: *Proc. of the 2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*. 2017, pp. 19–24.

[Fer+18] Francisco-Javier Ferrández-Pastor, Higinio Mora, Antonio Jimeno-Morenilla, and Bruno Volckaert. “Deployment of IoT edge and fog computing technologies to develop smart building services”. In: *Sustainability* 10.11 (2018), pp. 1–23.

[Fia+17] Claudio Fiandrino, Andrea Capponi, Giuseppe Cacciatore, Dzmitry Kliazovich, Ulrich Sorger, Pascal Bouvry, Burak Kantarci, Fabrizio Granelli, and Stefano Giordano. “CrowdSenSim: a Simulation Platform for Mobile Crowdsensing in Realistic Urban Environments”. In: *IEEE Access* 5 (2017), pp. 3490–3503.

[FJ10] Gaojun Fan and Shiyao Jin. “Coverage Problem in Wireless Sensor Network: A Survey”. In: *Journal of Networks* 5.9 (2010), pp. 1033–1040.

[Flo+15] Huber Flores, Pan Hui, Sasu Tarkoma, Yong Li, Satish Narayana Srirama, and Rajkumar Buyya. “Mobile code offloading: From concept to practice and beyond”. In: *IEEE Communications Magazine* 53.3 (2015), pp. 80–88.

[FLR13] Niroshinie Fernando, Seng Wai Loke, and J. Wenny Rahayu. “Mobile cloud computing: A survey”. In: *Future Generation Computer Systems* 29.1 (2013), pp. 84–106.

[Fow14] Martin Fowler. *Microservices - a definition of this new architectural term*. `https://martinfowler.com/articles/microservices.html`. Accessed: 2019-03-04. 2014.

[Fre+13] Sylvain Frey, Ada Diaconescu, David Menga, and Isabelle Demeure. “A holonic control architecture for a heterogeneous multi-objective smart micro-grid”. In: *Proc. of the IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*. 2013, pp. 21–30.

[Frö+16] Alexander Frömmgen, Jens Heuschkel, Patrick Jahnke, Fabio Cuozzo, Immanuel Schweizer, Patrick Eugster, Max Mühlhäuser, and Alejandro P. Buchmann. “Crowdsourcing Measurements of Mobile Network Performance and Mobility During a Large Scale Event”. In: *Proc. of the 17th International Passive and Active Measurement Conference (PAM)*. 2016, pp. 70–82.

[Fu+18] Jun-Song Fu, Yun Liu, Han-Chieh Chao, Bharat K Bhargava, and Zhen-Jiang Zhang. “Secure data storage and searching for industrial IoT by integrating fog computing and cloud computing”. In: *Transactions on Industrial Informatics* 14.10 (2018), pp. 4519–4528.

[Gao+17] Mingze Gao, Qian Wang, Md Tanvir Arafin, Yongqiang Lyu, and Gang Qu. “Approximate Computing for Low Power and Security in the Internet of Things”. In: *IEEE Computer* 50.6 (2017), pp. 27–34.

[Gau+13] E. I. Gaura, J. Brusey, M. Allen, R. Wilkins, D. Goldsmith, and R. Rednic. “Edge Mining the Internet of Things”. In: *IEEE Sensors Journal* 13.10 (2013), pp. 3816–3825.

[GC04] Sachin Goyal and John Carter. “A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices”. In: *Proc. of the 6th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*. 2004, pp. 186–195.

[GD08] Amitabha Amitava Ghosh and Sajal K. Das. “Coverage and connectivity issues in wireless sensor networks: A survey”. In: *Pervasive and Mobile Computing* 4.3 (2008), pp. 303–334.

[Ged+17] Julien Gedeon, Christian Meurisch, Disha Bhat, Michael Stein, Lin Wang, and Max Mühlhäuser. “Router-based Brokering for Surrogate Discovery in Edge Computing”. In: *Proc. of the International Conference on Distributed Computing Systems Workshops (ICDCS Workshops)*. 2017, pp. 145–150.

[Ged+18a] Julien Gedeon, Jens Heuschkel, Lin Wang, and Max Mühlhäuser. “Fog Computing: Current Research and Future Challenges”. In: *Proc. of 1.GI/ITG KuVS Fachgespräche Fog Computing*. 2018, pp. 1–4.

[Ged+18b] Julien Gedeon, Nicolás Himmelmann, Patrick Felka, Fabian Herrlich, Michael Stein, and Max Mühlhäuser. “vStore: A Context-Aware Framework for Mobile Micro-Storage at the Edge”. In: *Proc. of the International Conference on Mobile Computing, Applications and Services (MobiCASE)*. 2018, pp. 165–182.

[Ged+18c] Julien Gedeon, Jeff Krisztinkovics, Christian Meurisch, Michael Stein, Lin Wang, and Max Mühlhäuser. “A Multi-Cloudlet Infrastructure for Future Smart Cities: An Empirical Study”. In: *Proc. of the 1st International Workshop on Edge Systems, Analytics and Networking (EdgeSys)*. ACM. 2018, pp. 19–24.

[Ged+18d] Julien Gedeon, Michael Stein, Jeff Krisztinkovics, Patrick Felka, Katharina Keller, Christian Meurisch, Lin Wang, and Max Mühlhäuser. “From Cell Towers to Smart Street Lamps: Placing Cloudlets on Existing Urban Infrastructures”. In: *Proc. of the 2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE. 2018, pp. 187–202.

[Ged+18e] Julien Gedeon, Michael Stein, Lin Wang, and Max Mühlhäuser. “On Scalable In-Network Operator Placement for Edge Computing”. In: *Proc. of the 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2018, pp. 1–9.

[Ged+19a] Julien Gedeon, Florian Brandherm, Rolf Egert, Tim Grube, and Max Mühlhäuser. “What the Fog? Edge Computing Revisited: Promises, Applications and Future Challenges”. In: *IEEE Access* 7 (2019), pp. 152847–152878.

[Ged+19b] Julien Gedeon, Martin Wagner, Jens Heuschkel, Lin Wang, and Max Mühlhäuser. “A Microservice Store for Efficient Edge Offloading”. In: *Proc. of the IEEE Global Communications Conference (GLOBECOM)*. 2019, pp. 1–6.

[Ged+20] Julien Gedeon, Sebastian Zengerle, Sebastian Alles, Florian Brandherm, and Max Mühlhäuser. “Sunstone: Navigating the Way Through the Fog”. In: *Proc. of the International Conference on Fog and Edge Computing (ICFEC)*. 2020, to appear.

[Ged17] Julien Gedeon. “Edge Computing via Dynamic In-network Processing”. In: *International Conference on Networked Systems (Netsys’17): PhD Forum*. 2017, pp. 1–2.

[GES08] Pawel Garbacki, Dick H. J. Epema, and Maarten van Steen. “Brokerplacement in latency-aware peer-to-peer networks”. In: *Computer Networks* 52.8 (2008), pp. 1617–1633.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google file system”. In: *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP)*. 2003, pp. 29–43.

[Gha+16] B. Ghazal, K. ElKhatib, K. Chahine, and M. Kherfan. “Smart traffic light control system”. In: *Proc. of the 2016 Third International Conference on Electrical, Electronics, Computer Engineering and their Applications (EECEA)*. 2016, pp. 140–145.

[Gho+19] Sara Gholami, Alireza Goli, Cor-Paul Bezemer, and Hamzeh Khazaei. “A Framework for Satisfying the Performance Requirements of Containerized Software Systems Through Multi-Versioning”. In: *Proc. of the International Conference on Performance Engineering (ICPE)*. 2019, pp. 1–11.

[Gia+15] Tuan Nguyen Gia, Mingzhe Jiang, Amir-Mohammad Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. “Fog Computing in Healthcare Internet of Things: A Case Study on ECG Feature Extraction”. In: *Proc. of the 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. 2015, pp. 356–363.

[Gil16] James N Gilmore. “Everywear: The quantified self and wearable fitness technologies”. In: *New Media & Society* 18.11 (2016), pp. 2524–2539.

[GJM19] Georgia Gkioxari, Justin Johnson, and Jitendra Malik. “Mesh R-CNN”. In: *Proc. of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9784–9794.

[Gol+19] Morteza Golkarifard, Ji Yang, Zhanpeng Huang, Ali Movaghar, and Pan Hui. “Dandelion: A Unified Code Offloading System for Wearable Computing”. In: *IEEE Transactions on Mobile Computing* 18.3 (2019), pp. 546–559.

[Gor+12] Mark S. Gordon, Davoud Anoushe Jamshidi, Scott A. Mahlke, Zhuoqing Morley Mao, and Xu Chen. “COMET: Code Offload by Migrating Execution Transparently”. In: *Proc. of the 13 USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2012, pp. 93–106.

[Gov+14] Nithyashri Govindarajan, Yogesh Simmhan, Nitin Jamadagni, and Prasant Misra. “Event Processing Across Edge and the Cloud for Internet of Things Applications”. In: *Proc. of the 20th International Conference on Management of Data*. COMAD ’14. 2014, pp. 101–104.

[GR18] Harshit Gupta and Umakishore Ramachandran. “FogStore: A Geo-Distributed Key-Value Store Guaranteeing Low Latency for Strongly Consistent Access”. In: *Proc. of the 12th ACM International Conference on Distributed and Event-based Systems*. DEBS ’18. 2018, pp. 148–159.

[Gre+08] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. “The cost of a cloud: research problems in data center networks”. In: *ACM SIGCOMM Computer Communication Review* 39.1 (2008), pp. 68–73.

[GS15] Julien Gedeon and Immanuel Schweizer. “Understanding Spatial and Temporal Coverage in Participatory Sensor Networks”. In: *Proc. of the 40th IEEE Local Computer Networks Conference Workshops (LCN Workshops)*. 2015, pp. 699–707.

[Gub+13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. “Internet of Things (IoT): A vision, architectural elements, and future directions”. In: *Future Generation Computer Systems* 29.7 (2013), pp. 1645–1660.

[Guo+18] Peizhen Guo, Bo Hu, Rui Li, and Wenjun Hu. “FoggyCache: Cross-Device Approximate Computation Reuse”. In: *Proc. of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom)*. 2018, pp. 19–34.

[Gup+11] Vaibhav Gupta, Debabrata Mohapatra, Sang Phill Park, Anand Raghunathan, and Kaushik Roy. “IMPACT: imprecise adders for low-power approximate computing”. In: *Proc. of the 2011 International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE/ACM, 2011, pp. 409–414.

[GXR18] Harshit Gupta, Zhuangdi Xu, and Umakishore Ramachandran. “DataFog: Towards a Holistic Data Management Platform for the IoT Age at the Network Edge”. In: *Proc. of the USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. 2018, pp. 1–6.

[GZG13] Lin Gu, Deze Zeng, and Song Guo. “Vehicular cloud computing: A survey”. In: *Proc. of the Global Communications Conference Workshops (GLOBECOM Workshops)*. 2013, pp. 403–407.

[Ha+14] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. “Towards Wearable Cognitive Assistance”. In: *Proc. of the 12th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys ’14. 2014, pp. 68–81.

[HAB17] S. Hassan, N. Ali, and R. Bahsoon. “Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity”. In: *Proc.of the 2017 IEEE International Conference on Software Architecture (ICSA)*. 2017, pp. 1–10.

[Hal+08] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. “Dynamic Software Product Lines”. In: *Computer* 41.4 (2008), pp. 93–95.

[Han+17] Dong Han, Ye Yan, Tao Shu, Liuqing Yang, and Shuguang Cui. “Cognitive Context-Aware Distributed Storage Optimization in Mobile Cloud Computing: A Stable Matching Based Approach”. In: *Proc. of the 37th IEEE International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 594–604.

[Hao+17] Pengzhan Hao, Yongshu Bai, Xin Zhang, and Yifan Zhang. “Edgecourier: an edge-hosted personal service for low-bandwidth document synchronization in mobile cloud storage services”. In: *Proc. of the Second ACM/IEEE Symposium on Edge Computing (SEC)*. 2017, 7:1–7:14.

[Has+18] Najmul Hassan, Saira Gillani, Ejaz Ahmed, Ibrar Yaqoob, and Muhammad Imran. “The Role of Edge Computing in Internet of Things”. In: *IEEE Communications Magazine* 56.11 (2018), pp. 110–115.

[Hav11] Matti Haverila. “Mobile phone feature preferences, customer satisfaction and repurchase intent among male users”. In: *Australasian Marketing Journal (AMJ)* 19.4 (2011), pp. 238–246.

[Haw+14] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal. “NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC)”. In: *IEEE Network* 28.6 (2014), pp. 18–26.

[HB16] Sara Hassan and Rami Bahsoon. “Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap”. In: *Proc. of the IEEE International Conference on Services Computing (SCC)*. 2016, pp. 813–818.

[HB96] J. Hawkinson and T. Bates. *RFC1930: Guidelines for creation, selection, and registration of an Autonomous System (AS)*. Internet Requests for Comments. 1996. URL: `https://www.rfc-editor.org/info/rfc1930`.

[Hec+18] Melanie Heck, Janick Edinger, Dominik Schäfer, and Christian Becker. “IoT Applications in Fog and Edge Computing: Where Are We and Where Are We Going?” In: *Proc. of the 27th International Conference on Computer Communication and Networks (ICCCN) Workshops*. 2018, pp. 1–6.

[Hel+19] Joseph M. Hellerstein, Jose M. Faleiro, Joseph Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. “Serverless Computing: One Step Forward, Two Steps Back”. In: *Proc. of the 9th Biennial Conference on Innovative Data Systems Research (CIDR)*. 2019.

[Hel02] S. Helal. “Standards for service discovery and delivery”. In: *IEEE Pervasive Computing* 1.3 (2002), pp. 95–100.

[Hen+16] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. “Serverless Computation with OpenLambda”. In: *Proc. of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2016)*. 2016, pp. 1–7.

[Heo+14] Taewook Heo, Kwangsoo Kim, Hyunhak Kim, Changwon Lee, Jae Hong Ryu, Youn Taik Leem, Jong Arm Jun, Chulsik Pyo, Seung-Mok Yoo, and JeongGil Ko. “Escaping from ancient Rome! Applications and challenges for designing smart cities”. In: *Transactions on Emerging Telecommunications Technologies* 25.1 (2014), pp. 109–119.

[Heu+17] Jens Heuschkel, Michael Stein, Lin Wang, and Max Mühlhäuser. “Beyond the core: Enabling software-defined control at the network edge”. In: *Proc. of the 2017 International Conference on Networked Systems (NetSys)*. 2017, pp. 1–6.

[Heu+19] Jens Heuschkel, Philipp Thomasberger, Julien Gedeon, and Max Mühlhäuser. “VirtualStack: Green High Performance Network Protocol Processing Leveraging FPGAs”. In: *Proc. of the IEEE Global Communications Conference (GLOBECOM)*. 2019, pp. 1–6.

[Him17] Nicolás Himmelmann. “Context-Aware Virtual Storage Framework for Mobile Devices”. Bachelor’s Thesis. Technische Universität Darmstadt, Department of Computer Science, 2017.

[Hir+13] Martin Hirzel, Robert Soulé, Scott Schneider, Bugra Gedik, and Robert Grimm. “A catalog of stream processing optimizations”. In: *ACM Computing Surveys* 46.4 (2013), 46:1–46:34.

[HK06] Robert Hirschfeld and Katsuya Kawamura. “Dynamic service adaptation”. In: *Software Practice and Experience* 36.11-12 (2006), pp. 1115–1131.

[HL08] H. Hartenstein and L. P. Laberteaux. “A tutorial survey on vehicular ad hoc networks”. In: *IEEE Communications Magazine* 46.6 (2008), pp. 164–171.

[HL16] Zijiang Hao and Qun Li. “Poster Abstract: EdgeStore: Integrating Edge Computing into Cloud-Based Storage Systems”. In: *Proc. of the IEEE/ACM Symposium on Edge Computing (SEC)*. 2016, pp. 115–116.

[HO13] Jie Han and Michael Orshansky. “Approximate computing: An emerging paradigm for energy-efficient design”. In: *Proc. of the 18th IEEE European Test Symposium (ETS)*. 2013, pp. 1–6.

[Hon+13] Kirak Hong, David J. Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. “Mobile fog: a programming model for large-scale applications on the internet of things”. In: *Proc. of the 2nd ACM Workshop on Mobile Cloud Computing (MCC)*. 2013, pp. 15–20.

[Hou+16] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen. “Vehicular Fog Computing: A Viewpoint of Vehicles as the Infrastructures”. In: *IEEE Transactions on Vehicular Technology* 65.6 (2016), pp. 3860–3873.

[How+17] Shaun Howell, Yacine Rezgui, Jean-Laurent Hippolyte, Bejay Jayan, and Haijiang Li. “Towards the next generation of smart grids: Semantic and holonic multi-agent management of distributed energy resources”. In: *Renewable and Sustainable Energy Reviews* 77 (2017), pp. 193–214.

[HS02] Thomas Heimrich and Günther Specht. “Enhancing ECA Rules for Distributed Active Database Systems”. In: *Proc. of the Web, Web-Services, and Database Systems, NODe 2002 Web and Database-Related Workshops*. Vol. 2593. Lecture Notes in Computer Science. 2002, pp. 199–205.

[HT05] Chi-Fu Huang and Yu-Chee Tseng. “The Coverage Problem in a Wireless Sensor Network”. In: *Mobile Networks and Applications* 10.4 (2005), pp. 519–528.

[Hu+15a] Wenlu Hu, Brandon Amos, Zhuo Chen, Kiryong Ha, Wolfgang Richter, Padmanabhan Pillai, Benjamin Gilbert, Jan Harkes, and Mahadev Satyanarayanan. “The Case for Offload Shaping”. In: *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile)*. 2015, pp. 51–56.

[Hu+15b] Y.C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young. “Mobile Edge Computing: A key technology towards 5G”. In: *ETSI Whitepaper* (2015), pp. 1–28.

[Hu+16] Wenlu Hu, Ying Gao, Kiryong Ha, Junjue Wang, Brandon Amos, Zhuo Chen, Padmanabhan Pillai, and Mahadev Satyanarayanan. “Quantifying the Impact of Edge Computing on Mobile Applications”. In: *Proc. of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys)*. 2016, 5:1–5:8.

[Hu+17] Pengfei Hu, Huansheng Ning, Tie Qiu, Yanfei Zhang, and Xiong Luo. “Fog Computing Based Face Identification and Resolution Scheme in Internet of Things”. In: *IEEE Transactions on Industrial Informatics* 13.4 (2017), pp. 1910–1920.

[Hua+11] Y. Huang, Z. Luan, R. He, and D. Qian. “Operator Placement with QoS Constraints for Distributed Stream Processing”. In: *Proc. of the 7th International Conference on Network and Service Management (CNSM)*. 2011, pp. 309–315.

[Hua+14] Chun-Ying Huang, Cheng-Hsin Hsu, De-Yu Chen, and Kuan-Ta Chen. “Quantifying User Satisfaction in Mobile Cloud Games”. In: *Proc. of the Workshop on Mobile Video Delivery*. MoViD’14. 2014, pp. 1–6.

[Hur04] Chris Hurley. *WarDriving: drive, detect, defend: a guide to wireless security*. Syngress Publishing, 2004.

[HV19] Cheol-Ho Hong and Blesson Varghese. “Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms”. In: *ACM Computing Surveys* 52.5 (2019), 97:1–97:37.

[IBD15] Stepan Ivanov, Kriti Bhargava, and William Donnelly. “Precision Farming: Sensor Analytics”. In: *IEEE Intelligent Systems* 30.4 (2015), pp. 76–80.

[Jal+16] Fatemeh Jalali, Kerry Hinton, Robert Ayre, Tansu Alpcan, and Rodney S. Tucker. “Fog Computing May Help to Save Energy in Cloud Computing”. In: *IEEE Journal on Selected Areas in Communications* 34.5 (2016), pp. 1728–1739.

[Jam+18] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov. “Microservices: The Journey So Far and Challenges Ahead”. In: *IEEE Software* 35.3 (2018), pp. 24–35.

[Jar+03] Amit P. Jardosh, Elizabeth M. Belding-Royer, Kevin C. Almeroth, and Subhash Suri. “Towards realistic mobility models for mobile ad hoc networks”. In: *Proc. of the Ninth Annual International Conference on Mobile Computing and Networking (MOBICOM)*. Ed. by David B. Johnson, Anthony D. Joseph, and Nitin H. Vaidya. 2003, pp. 217–229.

[Jay+14] Prem Prakash Jayaraman, João Bártolo Gomes, Hai-Long Nguyen, Zahraa Said Abdallah, Shonali Krishnaswamy, and Arkady B. Zaslavsky. “CARDAP: A Scalable Energy-Efficient Context Aware Distributed Mobile Data Analytics Platform for the Fog”. In: *Proc. Advances in Databases and Information Systems (ADBIS)*. 2014, pp. 192–206.

[JCL17] M. Jia, J. Cao, and W. Liang. “Optimal Cloudlet Placement and User to Cloudlet Allocation in Wireless Metropolitan Area Networks”. In: *IEEE Transactions on Cloud Computing* 5.4 (2017), pp. 725–737.

[Ji+12] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li. “Big Data Processing in Cloud Computing Environments”. In: *Proc. of the 2012 12th International Symposium on Pervasive Systems, Algorithms and Networks*. 2012, pp. 17–23.

[Jia+18] G. Jia, G. Han, A. Li, and J. Du. “SSL: Smart Street Lamp Based on Fog Computing for Smarter Cities”. In: *IEEE Transactions on Industrial Informatics* 14.11 (2018), pp. 4995–5004.

[Jia+19] Congfeng Jiang, Xiaolan Cheng, Honghao Gao, Xin Zhou, and Jian Wan. “Toward Computation Offloading in Edge Computing: A Survey”. In: *IEEE Access* 7 (2019), pp. 131543–131558.

[JLR13] Aleksandar Jovicic, Junyi Li, and Tom Richardson. “Visible light communication: opportunities, challenges and the path to market”. In: *IEEE Communications Magazine* 51.12 (2013), pp. 26–32.

[Jon+17] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. “Occupy the Cloud: Distributed Computing for the 99%”. In: *Proc. of the 2017 Symposium on Cloud Computing*. SoCC ’17. ACM, 2017, pp. 445–451.

[JSK18a] S. Jeong, O. Simeone, and J. Kang. “Mobile Edge Computing via a UAV-Mounted Cloudlet: Optimization of Bit Allocation and Path Planning”. In: *IEEE Transactions on Vehicular Technology* 67.3 (2018), pp. 2049–2063.

[JSK18b] Seongah Jeong, Osvaldo Simeone, and Joonhyuk Kang. “Mobile Edge Computing via a UAV-Mounted Cloudlet: Optimization of Bit Allocation and Path Planning”. In: *IEEE Transactions on Vehicular Technology* 67.3 (2018), pp. 2049–2063.

[JV01] Kamal Jain and Vijay V. Vazirani. “Approximation algorithms for metric facility location and *k*-Median problems using the primal-dual schema and Lagrangian relaxation”. In: *Journal of the ACM* 48.2 (2001), pp. 274–296.

[KAB13] Ayat Khairy, Hany H. Ammar, and Reem Bahgat. “Smartphone Energizer: Extending Smartphone’s battery life with smart offloading”. In: *Proc. of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2013, pp. 329–336.

[Kad+14] K. Kadir, M. K. Kamaruddin, H. Nasir, S. I. Safie, and Z. A. K. Bakti. “A comparative study between LBP and Haar-like features for Face Detection using OpenCV”. In: *Proc. of the 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)*. 2014, pp. 335–339.

[Käm+14] T. Kämäräinen, M. Siekkinen, Y. Xiao, and A. Ylä-Jääski. “Towards pervasive and mobile gaming with distributed cloud infrastructure”. In: *Proc. of the 2014 13th Annual Workshop on Network and Systems Support for Games*. 2014, pp. 1–6.

[Kan+19] Ram Srivatsa Kannan, Lavanya Subramanian, Ashwin Raju, Jeongseob Ahn, Jason Mars, and Lingjia Tang. “GrandSLAm: Guaranteeing SLAs for Jobs in Microservices Execution Frameworks”. In: *Proc. of the 14th EuroSys Conference*. 2019, 34:1–34:16.

[KB10] Mads Darø Kristensen and Niels Olof Bouvin. “Scheduling and development support in the Scavenger cyber foraging system”. In: *Pervasive and Mobile Computing* 6.6 (2010), pp. 677–692.

[Kem+10] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri E. Bal. “Cuckoo: A Computation Offloading Framework for Smartphones”. In: *Proc. of the 2nd International Conference on Mobile Computing, Applications and Services (MobiCASE)*. 2010, pp. 59–79.

[KJP15] A. Krylovskiy, M. Jahn, and E. Patti. “Designing a Smart City Internet of Things Platform with Microservice Architecture”. In: *Proc. of the 2015 3rd International Conference on Future Internet of Things and Cloud*. 2015, pp. 25–30.

[KL10] Karthik Kumar and Yung-Hsiang Lu. “Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?” In: *IEEE Computer* 43.4 (2010), pp. 51–56.

[Kli+18] Ana Klimovic, Yawen Wang, Christos Kozyrakis, Patrick Stuedi, Jonas Pfefferle, and Animesh Trivedi. "Understanding Ephemeral Storage for Serverless Analytics". In: *Proc. of the 2018 USENIX Annual Technical Conference (USENIX ATC)*. 2018, pp. 789–794.

[KLT16] H. Kang, M. Le, and S. Tao. "Container and Microservice Driven Design for Cloud Infrastructure DevOps". In: *Proc. of the 2016 IEEE International Conference on Cloud Engineering (IC2E)*. 2016, pp. 202–211.

[Kon+06] Woralak Kongdenfha, Régis Saint-Paul, Boualem Benatallah, and Fabio Casati. "An Aspect-Oriented Framework for Service Adaptation". In: *Proc. of the 4th International Conference on Service-Oriented Computing (ICSOC)*. 2006, pp. 15–26.

[Kos+12] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading". In: *Proc. of the IEEE Conference on Computer Communications (INFOCOM)*. 2012, pp. 945–953.

[Kra+17] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma. "Fog Computing in Healthcare–A Review and Discussion". In: *IEEE Access* 5 (2017), pp. 9206–9222.

[Kre+15] Diego Kreutz, Fernando M. V. Ramos, Paulo Jorge Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76.

[Kri19] Jeff Krisztinkovics. "Cloudlet-Abdeckung im urbanan Raum". Bachelor's Thesis. Technische Universität Darmstadt, Department of Computer Science, 2019.

[Kum+13] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat K. Bhargava. "A Survey of Computation Offloading for Mobile Systems". In: *Mobile Networks and Applications* 18.1 (2013), pp. 129–140.

[KYK12] Dejan Kovachev, Tian Yu, and Ralf Klamma. "Adaptive Computation Offloading from Mobile Devices into the Cloud". In: *Proc. of the 10th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*. 2012, pp. 784–791.

[Lai+17] Zeqi Lai, Y. Charlie Hu, Yong Cui, Linhui Sun, and Ningwei Dai. "Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices". In: *Proc. of the 23rd Annual International Conference on Mobile Computing and Networking*. MobiCom '17. 2017, pp. 409–421.

[LC11] Changlei Liu and Guohong Cao. "Spatial-Temporal Coverage Optimization in Wireless Sensor Networks". In: *IEEE Transactions on Mobile Computing* 10.4 (2011), pp. 465–478.

[Lee+13] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong. "Mobile Data Offloading: How Much Can WiFi Deliver?" In: *IEEE/ACM Transactions on Networking* 21.2 (2013), pp. 536–550.

[Lee+16] Eun-Kyu Lee, Mario Gerla, Giovanni Pau, Uichin Lee, and Jae-Han Lim. "Internet of Vehicles: From intelligent grid to autonomous cars and vehicular fogs". In: *International Journal of Distributed Sensor Networks* 12.9 (2016), pp. 1–14.

[Lew+14] G. Lewis, S. Echeverría, S. Simanta, B. Bradshaw, and J. Root. "Tactical Cloudlets: Moving Cloud Computing to the Edge". In: *Proc. of the 2014 IEEE Military Communications Conference*. 2014, pp. 1440–1446.

[LGS17] Prasanth Lade, Rumi Ghosh, and Soundar Srinivasan. "Manufacturing analytics and industrial internet of things". In: *Intelligent Systems* 32.3 (2017), pp. 74–79.

[LH18] Qiang Liu and Tao Han. "DARE: Dynamic Adaptive Mobile Augmented Reality with Edge Computing". In: *Proc. of the 2018 IEEE 26th International Conference on Network Protocols (ICNP)*. 2018, pp. 1–11.

[Li+11] M. Li, W. Cheng, K. Liu, Y. He, X. Li, and X. Liao. "Sweep Coverage with Mobile Sensors". In: *IEEE Transactions on Mobile Computing* 10.11 (2011), pp. 1534–1545.

[Li+15] Jiwei Li, Zhe Peng, Bin Xiao, and Yu Hua. "Make smartphones last a day: Pre-processing based computer vision application offloading". In: *Proc. of the 12th Annual IEEE International Conference on Sensing, Communication (SECON)*. 2015, pp. 462–470.

[Li+16a] D. Li, T. Salonidis, N. V. Desai, and M. C. Chuah. "DeepCham: Collaborative Edge-Mediated Adaptive Deep Learning for Mobile Object Recognition". In: *Proc. of the 2016 IEEE/ACM Symposium on Edge Computing (SEC)*. 2016, pp. 64–76.

[Li+16b] Hongxing Li, Guochu Shou, Yihong Hu, and Zhigang Guo. "Mobile Edge Computing: Progress and Challenges". In: *Proc. of the 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. 2016, pp. 83–84.

[Li+18] Chao Li, Yushu Xue, Jing Wang, Weigong Zhang, and Tao Li. "Edge-Oriented Computing Paradigms: A Survey on Architecture Design and System Management". In: *ACM Computing Surveys* 51.2 (2018), 39:1–39:34.

[Lia+11] L. Liao, W. Chen, C. Zhang, L. Zhang, D. Xuan, and W. Jia. "Two Birds With One Stone: Wireless Access Point Deployment for Both Coverage and Localization". In: *IEEE Transactions on Vehicular Technology* 60.5 (2011), pp. 2239–2252.

[Lia+18] Konstantinos Liakos, Patrizia Busato, Dimitrios Moshou, Simon Pearson, and Dionysis Bochtis. "Machine Learning in Agriculture: A Review". In: *Sensors* 18.8 (2018), pp. 1–29.

[Liu+08] Benyuan Liu, Olivier Dousse, Jie Wang, and Anwar Saipulla. "Strong Barrier Coverage of Wireless Sensor Networks". In: *Proc. of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. MobiHoc. 2008, pp. 411–420.

[Liu+17] Yang Liu, Changqiao Xu, Yufeng Zhan, Zhixin Liu, Jianfeng Guan, and Hongke Zhang. "Incentive mechanism for computation offloading using edge computing: A Stackelberg game approach". In: *Computer Networks* 129 (2017), pp. 399–409.

[Liu+18] Liangkai Liu, Xingzhou Zhang, Mu Qiao, and Weisong Shi. "SafeShareRide: Edge-Based Attack Detection in Ridesharing Services". In: *Proc. of the 2018 IEEE/ACM Symposium on Edge Computing (SEC)*. 2018, pp. 17–29.

[LLS08] Geetika T. Lakshmanan, Ying Li, and Rob Strom. "Placement strategies for internet-scale data stream systems". In: *IEEE Internet Computing* 12.6 (2008), pp. 50–60.

[LMB17] Ivan Lujic, Vincenzo De Maio, and Ivona Brandic. "Efficient Edge Storage Management Based on Near Real-Time Forecasts". In: *Proc. of the 1st IEEE International Conference on Fog and Edge Computing (ICFEC)*. 2017, pp. 21–30.

[Loc+08] Christian Lochert, Björn Scheuermann, Christian Wewetzer, Andreas Lübke, and Martin Mauve. "Data Aggregation and Roadside Unit Placement for a VANET Traffic Information System". In: *Proc. of the 5th International Workshop on Vehicular Ad Hoc Networks (VANET)*. 2008, pp. 58–65.

[LQB18] Peng Liu, Bozhao Qi, and Suman Banerjee. "EdgeEye: An edge service framework for real-time intelligent video analytics". In: *Proc. of the 1st International Workshop on Edge Systems, Analytics and Networking (EdgeSys)*. 2018, pp. 1–6.

[LR17] Maroš Lacinák and Jozef Ristvej. "Smart City, Safety and Security". In: *Procedia Engineering* 192 (2017), pp. 522–527.

[LS17] Yuhua Lin and Haiying Shen. "CloudFog: Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Service". In: *IEEE Transactions on Parallel and Distributes Systems* 28.2 (2017), pp. 431–445.

[Lua+16] Tom H Luan, Longxiang Gao, Zhi Li, Yang Xiang, Guiyi We, and Limin Sun. "A view of fog computing from networking perspective". In: *CoRR* abs/1602.01509 (2016).

[Lui+15] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary. "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions". In: *Proc. of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, pp. 98–106.

[LWB16] Peng Liu, Dale Willis, and Suman Banerjee. "ParaDrop: Enabling Lightweight Multi-tenancy at the Network's Extreme Edge". In: *Proc. of the IEEE/ACM Symposium on Edge Computing (SEC)*. 2016, pp. 1–13.

[LWX01] Zhiyuan Li, Cheng Wang, and Rong Xu. "Computation offloading to save energy on handheld devices: a partition scheme". In: *Proc. of the 2001 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. 2001, pp. 238–246.

[LY18] Chun-Cheng Lin and Jhih-Wun Yang. “Cost-efficient deployment of fog computing systems at logistics centers in industry 4.0”. In: *Transactions on Industrial Informatics* 14.10 (2018), pp. 4603–4611.

[LYS16] Jiayi Liu, Qinghai Yang, and Gwendal Simon. “Optimal and Practical Algorithms for Implementing Wireless CDN Based on Base Stations”. In: *Proc. of the IEEE 83rd Vehicular Technology Conference (VTC)*. 2016, pp. 1–5.

[LZC18] En Li, Zhi Zhou, and Xu Chen. “Edge Intelligence: On-Demand Deep Learning Model Co-Inference with Device-Edge Synergy”. In: *Proc. of the 2018 Workshop on Mobile Edge Communications*. MECOMM’18. 2018, pp. 31–36.

[Ma+17] Longjie Ma, Jigang Wu, Long Chen, and Zhusong Liu. “Fast algorithms for capacitated cloudlet placements”. In: *Proc. of the 21st IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. 2017, pp. 439–444.

[MA10] Raymond Mulligan and Habib M. Ammari. “Coverage in Wireless Sensor Networks: A Survey”. In: *Network Protocols & Algorithms* 2.2 (2010), pp. 27–53.

[Mad+13] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. “Unikernels: Library Operating Systems for the Cloud”. In: *Proc. of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’13. 2013, pp. 461–472.

[Mai+09] Nicolas Maisonneuve, Matthias Stevens, Maria E. Niessen, and Luc Steels. “NoiseTube: Measuring and mapping noise pollution with mobile phones”. In: *Proc. of the 4th International ICSC Symposium*. 2009, pp. 215–228.

[Man+04] Katerina Mania, Bernard D. Adelstein, Stephen R. Ellis, and Michael I. Hill. “Perceptual Sensitivity to Head Tracking Latency in Virtual Environments with Varying Degrees of Scene Complexity”. In: *Proc. of the 1st Symposium on Applied Perception in Graphics and Visualization*. APGV ’04. 2004, pp. 39–47.

[Man+17] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. “My VM is Lighter (and Safer) than your Container”. In: *Proc. of the 26th Symposium on Operating Systems Principles (SOSP)*. 2017, pp. 218–233.

[Man+18] Johannes Manner, Martin EndreB, Tobias Heckel, and Guido Wirtz. “Cold Start Influencing Factors in Function as a Service”. In: *Proc. of the 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC)*. 2018, pp. 181–188.

[Mao+17] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled Ben Letaief. “A Survey on Mobile Edge Computing: The Communication Perspective”. In: *IEEE Communications Surveys and Tutorials* 19.4 (2017), pp. 2322–2358.

[Mar+14] João Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Andrei Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. "ClickOS and the Art of Network Function Virtualization". In: *Proc. of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2014, pp. 459–473.

[Mar+17] Eva Marín-Tordera, Xavier Masip-Bruin, Jordi Garcia Almiñana, Admela Jukan, Guang-Jie Ren, and Jiafeng Zhu. "Do we all really know what a fog node is? Current trends towards an open definition". In: *Computer Communications* 109 (2017), pp. 117–130.

[Mar+19] Karola Marky, Andreas Weiß, Julien Gedeon, and Sebastian Günther. "Mastering Music Instruments through Technology in Solo Learning Sessions". In: *Proc. of the 7th Workshop on Interacting with Smart Objects (SmartObjects '19)*. 2019, pp. 1–6.

[May+17] Ruben Mayer, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. "FogStore: Toward a distributed data store for Fog computing". In: *IEEE Fog World Congress*. 2017, pp. 1–6.

[MAŽ18] M. Marjanović, A. Antonić, and I. P. Žarko. "Edge Computing Architecture for Mobile Crowdsensing". In: *IEEE Access* 6 (2018), pp. 10662–10674.

[MB17] Pavel Mach and Zdenek Becvar. "Mobile Edge Computing: A Survey on Architecture and Computation Offloading". In: *IEEE Communications Surveys and Tutorials* 19.3 (2017), pp. 1628–1656.

[McG+02] John D. McGregor, Linda M. Northrop, Salah Jarrad, and Klaus Pohl. "Guest Editors' Introduction: Initiating Software Product Lines". In: *IEEE Software* 19.4 (2002), pp. 24–27.

[Med+15] Alexey Medvedev, Petr Fedchenkov, Arkady Zaslavsky, Theodoros Anagnostopoulos, and Sergey Khoruzhnikov. "Waste Management as an IoT-Enabled Service in Smart Cities". In: *Proc. Internet of Things, Smart Spaces, and Next Generation Networks and Systems - 15th International Conference, NEW2AN 2015, and 8th Conference, ruSMART 2015*. 2015, pp. 104–115.

[Men+17] Hamid Menouar, Ismail Güvenç, Kemal Akkaya, A. Selcuk Uluagac, Abdullah Kadri, and Adem Tuncer. "UAV-Enabled Intelligent Transportation Systems for the Smart City: Applications and Challenges". In: *IEEE Communications Magazine* 55.3 (2017), pp. 22–28.

[Men+18] Nabor C. Mendonça, David Garlan, Bradley R. Schmerl, and Javier Cámara. "Generality vs. reusability in architecture-based self-adaptation: The case for self-adaptive microservices". In: *Proc. of the 12th European Conference on Software Architecture: Companion Proceedings, (ECSA)*. 2018, 18:1–18:6.

[Meu+15] Christian Meurisch, Alexander Seeliger, Benedikt Schmidt, Immanuel Schweizer, Fabian Kaup, and Max Mühlhäuser. "Upgrading wireless home routers for enabling large-scale deployment of cloudlets". In: *Proc. of the 7th International Conference on Mobile Computing, Applications, and Services (MobiCASE)*. 2015, pp. 12–29.

[Meu+17a] Christian Meurisch, Julien Gedeon, Artur Gogel, The An Binh Nguyen, Fabian Kaup, Florian Kohnhäuser, Lars Baumgärtner, Milan Schmittner, and Max Mühlhäuser. “Temporal Coverage Analysis of Router-Based Cloudlets Using Human Mobility Patterns”. In: *Proc. of the IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2017, pp. 1–6.

[Meu+17b] Christian Meurisch, Julien Gedeon, The An Binh Nguyen, Fabian Kaup, and Max Mühlhäuser. “Decision Support for Computational Offloading by Probing Unknown Services”. In: *Proc. of the 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2017, pp. 1–9.

[Meu+17c] Christian Meurisch, The An Binh Nguyen, Julien Gedeon, Florian Kohnhäuser, Milan Schmittner, Stefan Niemczyk, Stefan Wullkotte, and Max Mühlhäuser. “Upgrading Wireless Home Routers as Emergency Cloudlet and Secure DTN Communication Bridge”. In: *Proc. of the 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2017, pp. 1–2.

[MFH19] Cristina Mihale-Wilson, Patrick Felka, and Oliver Hinz. “The Bright and the Dark Side of Smart Lights? The Protective Effect of Smart City Infrastructures”. In: *Proc. of the Hawaii International Conference on System Sciences (HICSS)*. 2019, pp. 3345–3354.

[MFP20] Octavian Machidon, Tine Fajfar, and Veljko Pejović. “Implementing Approximate Mobile Computing”. In: *Proc. of the 2020 Workshop on Approximate Computing Across the Stack (WAX)*. 2020, pp. 1–3.

[MHS17] Shaimaa M. Mohamed, Haitham S. Hamza, and Iman Aly Saroit. “Coverage in mobile wireless sensor networks (M-WSN): A survey”. In: *Computer Communications* 110 (2017), pp. 133–150.

[Mij+16] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. “Network Function Virtualization: State-of-the-Art and Research Challenges”. In: *IEEE Communications Surveys Tutorials* 18.1 (2016), pp. 236–262.

[Mit16] Sparsh Mittal. “A Survey of Techniques for Approximate Computing”. In: *ACM Computing Surveys* 48.4 (2016), 62:1–62:33.

[MJ16] Kianoosh Mokhtarian and Hans-Arno Jacobsen. “Coordinated caching in planet-scale CDNs: Analysis of feasibility and benefits”. In: *Proc. of the IEEE Conference on Computer Communications (INFOCOM)*. 2016, pp. 1–9.

[MK16] N. Mohan and J. Kangasharju. “Edge-Fog cloud: A distributed cloud for Internet of Things computations”. In: *Proc. of the 2016 Cloudification of the Internet of Things (CIoT)*. 2016, pp. 1–6.

[MKB18a] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. “Fog Computing: A Taxonomy, Survey and Future Directions”. In: *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*. Ed. by Beniamino Di Martino, Kuan-Ching Li, Laurence T. Yang, and Antonio Esposito. Springer Singapore, 2018, pp. 103–130.

[MKB18b] F. Messaoudi, A. Ksentini, and P. Bertin. “Toward a Mobile Gaming Based-Computation Offloading”. In: *Proc. of the 2018 IEEE International Conference on Communications (ICC)*. 2018, pp. 1–7.

[MN10] Antti P. Miettinen and Jukka K. Nurminen. “Energy Efficiency of Mobile Clients in Cloud Computing”. In: *Proc. of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. 2010.

[Moh+11] Debabrata Mohapatra, Vinay K. Chippa, Anand Raghunathan, and Kaushik Roy. “Design of voltage-scalable meta-functions for approximate computing”. In: *Proc. of the 2011 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2011, pp. 950–955.

[Moh+18] Nitinder Mohan, Aleksandr Zavodovski, Pengyuan Zhou, and Jussi Kangasharju. “Anveshak: Placing Edge Servers In The Wild”. In: *Proc. of the 2018 Workshop on Mobile Edge Communications (MECOMM)*. 2018, pp. 7–12.

[Mor+16] Richard Mortier, Jianxin Zhao, Jon Crowcroft, Liang Wang, Qi Li, Hamed Haddadi, Yousef Amar, Andy Crabtree, James Colley, Tom Lodge, et al. “Personal data management with the databox: What’s inside the box?” In: *Proc. of the 2016 ACM Workshop on Cloud-Assisted Networking*. 2016, pp. 49–54.

[Mor+17] Seyed Hossein Mortazavi, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips, and Eyal de Lara. “Cloudpath: a multi-tier cloud computing framework”. In: *Proc. of the Second ACM/IEEE Symposium on Edge Computing (SEC)*. 2017, 20:1–20:13.

[Mor+18a] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jörg Ott. “Consolidate IoT Edge Computing with Lightweight Virtualization”. In: *IEEE Network* 32.1 (2018), pp. 102–111.

[Mor+18b] Thierry Moreau, Joshua San Miguel, Mark Wyse, James Bornholt, Armin Alaghi, Luis Ceze, Natalie D. Enright Jerger, and Adrian Sampson. “A Taxonomy of General Purpose Approximate Computing Techniques”. In: *Embedded Systems Letters* 10.1 (2018), pp. 2–5.

[Mor+18c] Seyed Hossein Mortazavi, Bharath Balasubramanian, Eyal de Lara, and Shankaranarayanan Puzhavakath Narayanan. “Pathstore, A Data Storage Layer For The Edge”. In: *Proc. of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys ’18. ACM, 2018, pp. 519–519.

[Mor+18d] Seyed Hossein Mortazavi, Bharath Balasubramanian, Eyal de Lara, and Shankaranarayanan Puzhavakath Narayanan. “Toward Session Consistency for the Edge”. In: *Proc. of the USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. 2018, pp. 1–6.

[Mot+13] Vinícius F. S. Mota, Daniel F. Macedo, Yacine Ghamri-Doudane, and José Marcos S. Nogueira. “On the feasibility of WiFi offloading in urban areas: The Paris case study”. In: *Proc. of the IFIP Wireless Days (WD)*. 2013, pp. 1–6.

[MPP10] Lefteris Mamatas, Ioannis Psaras, and George Pavlou. “Incentives and Algorithms for Broadband Access Sharing”. In: *Proc. of the 2010 ACM SIGCOMM Workshop on Home Networks*. HomeNets ’10. 2010, pp. 19–24.

[MPZ10] X. Meng, V. Pappas, and L. Zhang. “Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement”. In: *Proc. of the 29th IEEE International Conference on Computer Communications (INFOCOM)*. 2010, pp. 1154–1162.

[MRD08] Oliver Moser, Florian Rosenberg, and Schahram Dustdar. “Non-Intrusive Monitoring and Service Adaptation for WS-BPEL”. In: *Proc. of the 17th International Conference on World Wide Web*. WWW ’08. 2008, pp. 815–824.

[MRS19] Sumit Kumar Monga, Sheshadri K. R, and Yogesh Simmhan. “Elf-Store: A Resilient Data Storage Service for Federated Edge and Fog Resources”. In: *Proc. of the 2019 IEEE International Conference on Web Services (ICWS)*. 2019, pp. 336–345.

[MS13] Anil Madhavapeddy and David J. Scott. “Unikernels: Rise of the Virtual Library Operating System”. In: *Queue* 11.11 (2013), 30:30–30:44.

[MSC15] Thierry Moreau, Adrian Sampson, and Luis Ceze. “Approximate Computing: Making Mobile Systems More Efficient”. In: *IEEE Pervasive Computing* 14.2 (2015), pp. 9–13.

[MSM17] Hassnaa Moustafa, Eve M. Schooler, and Jessica McCarthy. “Reverse CDN in Fog Computing: The lifecycle of video data in connected and autonomous vehicles”. In: *Proc. of the IEEE Fog World Congress (FWC)*. 2017, pp. 1–5.

[MTC17] D. Mazza, D. Tarchi, and G. E. Corazza. “A Unified Urban Mobile Cloud Computing Offloading Mechanism for Smart Cities”. In: *IEEE Communications Magazine* 55.3 (2017), pp. 30–37.

[Muk+17] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. A. Ferrag, N. Choudhury, and V. Kumar. “Security and Privacy in Fog Computing: Challenges”. In: *IEEE Access* 5 (2017), pp. 19293–19304.

[Mül+17] Florian Müller, Sebastian Günther, Azita Hosseini Nejad, Niloofar Dezfuli, Mohammadreza Khalilbeigi, and Max Mühlhäuser. “Cloudbits: supporting conversations through augmented zero-query search visualization”. In: *Proc. of the 5th Symposium on Spatial User Interaction (SUI)*. 2017, pp. 30–38.

[Nan+17] Yucen Nan, Wei Li, Wei Bao, Flávia Coimbra Delicato, Paulo F. Pires, Yong Dou, and Albert Y. Zomaya. “Adaptive Energy-Aware Computation Offloading for Cloud of Things Systems”. In: *IEEE Access* 5 (2017), pp. 23947–23957.

[Nar+19] Matteo Nardelli, Valeria Cardellini, Vincenzo Grassi, and Francesco Lo Presti. “Efficient Operator Placement for Distributed Data Stream Processing Applications”. In: *IEEE Transactions on Parallel and Distributed Systems* 30.8 (2019), pp. 1753–1767.

[NB12] Ebisa Negeri and Nico Baken. “Architecting the smart grid as a holarchy”. In: *Proc. of the 1st International Conference on Smart Grids and Green IT Systems*. 2012, pp. 1–6.

[Ngu09] Thanh Nguyen. “Indexing PostGIS databases and spatial Query performance evaluations”. In: *International Journal of Geoinformatics* 5 (2009), pp. 1–9.

[NS14] Dawn Nafus and Jamie Sherman. “Big data, big questions| this one does not go up to 11: the quantified self movement as an alternative big data practice”. In: *International journal of communication* 8 (2014), pp. 1784–1794.

[Nun+15] Swaroop Nunna, Apostolos Kousaridas, Mohamed Ibrahim, Markus Dillinger, Christoph Thuemmler, Hubertus Feussner, and Armin Schneider. “Enabling Real-Time Context-Aware Collaboration through 5G and Mobile Edge Computing”. In: *Proc. of the 2015 12th International Conference on Information Technology - New Generations*. 2015, pp. 601–605.

[Nuo+06] Teemu Nuortio, Jari Kytöjoki, Harri Niska, and Olli Bräysy. “Improved route planning and scheduling of waste collection and transport”. In: *Expert Systems with Applications* 30.2 (2006), pp. 223–232.

[OBL16] Gabriel Orsini, Dirk Bade, and Winfried Lamersdorf. “CloudAware: A Context-Adaptive Middleware for Mobile Edge and Cloud Computing Applications”. In: *Proc. of the 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2016, pp. 216–221.

[Ola+12] Rafael Olaechea, Steven Stewart, Krzysztof Czarnecki, and Derek Rayside. “Modelling and Multi-Objective Optimization of Quality Attributes in Variability-Rich Software”. In: *Proc. of the 4th International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages*. NFPinDSML ’12. 2012, pp. 1–6.

[Oye17] Emmanuel Oyekanlu. “Predictive edge computing for time series of industrial IoT and large scale critical infrastructure based on open-source software analytic of big data”. In: *Proc. of the 2017 IEEE International Conference on Big Data (Big Data)*. 2017, pp. 1663–1669.

[PA97] Daniel A. Peak and M. H. Azadmanesh. “Centralization/decentralization cycles in computing: Market evidence”. In: *Information & Management* 31.6 (1997), pp. 303–317.

[Pah+16] Claus Pahl, Sven Helmer, Lorenzo Miori, Julian Sanin, and Brian Lee. “A Container-Based Edge Cloud PaaS Architecture Based on RaspberryPi Clusters”. In: *Proc. of the 4th IEEE International Conference on Future Internet of Things and Cloud Workshops (FiCloud)*. 2016, pp. 117–124.

[Pan+13a] Gang Pan, Guande Qi, Wangsheng Zhang, Shijian Li, Zhaohui Wu, and Laurence Tianruo Yang. “Trace Analysis and Mining for Smart Cities: Issues, Methods, and Applications”. In: *IEEE Communications Magazine* 51.6 (2013).

[Pan+13b] R. K. Panta, R. Jana, F. Cheng, Y. R. Chen, and V. A. Vaishampayan. “Phoenix: Storage Using an Autonomous Mobile Infrastructure”. In: *IEEE Transactions on Parallel and Distributed Systems* 24.9 (2013), pp. 1863–1873.

[Pan+15] Zhengyuan Pang, Lifeng Sun, Zhi Wang, Erfang Tian, and Shiqiang Yang. “A Survey of Cloudlet Based Mobile Computing”. In: *Proc. of the International Conference on Cloud Computing and Big Data (CCBD)*. 2015, pp. 268–275.

[Pan+16] Jianli Pan, Lin Ma, Ravishankar Ravindran, and Peyman TalebiFard. “HomeCloud: An edge cloud framework and testbed for new application delivery”. In: *Proc. of the 23rd International Conference on Telecommunications (ICT)*. 2016, pp. 1–6.

[Pap03] Mike P. Papazoglou. “Service-Oriented Computing: Concepts, Characteristics and Directions”. In: *Proc. of the Fourth International Conference on Web Information Systems Engineering*. WISE ’03. 2003, pp. 3–12.

[Par+14] Jongse Park, Xin Zhang, Kangqi Ni, Hadi Esmaeilzadeh, and Mayur Naik. *Expax: A framework for automating approximate programming*. Tech. rep. Georgia Institute of Technology, 2014, pp. 1–17.

[Pas+18] Francisco Javier Ferrández Pastor, Juan Manuel García Chamizo, Mario Nieto-Hidalgo, and José Mora-Martínez. “Precision Agriculture Design Method Using a Distributed Computing Architecture on Internet of Things Context”. In: *Sensors* 18.6 (2018), pp. 1–21.

[PBC17] Juan F. Pérez, Robert Birke, and Lydia Y. Chen. “On the latency-accuracy tradeoff in approximate MapReduce jobs”. In: *Proc. of the 2017 IEEE Conference on Computer Communications (INFOCOM)*. 2017, pp. 1–9.

[PD99] Norman W. Paton and Oscar Díaz. “Active Database Systems”. In: *ACM Computing Surveys* 31.1 (1999), pp. 63–103.

[Pej18] Veljko Pejovic. “Towards Approximate Mobile Computing”. In: *GetMobile: Mobile Computing and Communications* 22.4 (2018), pp. 9–12.

[Pen+15] Boyang Peng, Mohammad Hosseini, Zhihao Hong, Reza Farivar, and Roy Campbell. “R-Storm: Resource-Aware Scheduling in Storm”. In: *Proc. of the 16th ACM/IFIP/USENIX Annual Middleware Conference (Middleware)*. 2015, pp. 149–161.

[Per+14a] Charith Perera, Arkady B. Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. “Context Aware Computing for The Internet of Things: A Survey”. In: *IEEE Communications Surveys & Tutorials* 16.1 (2014), pp. 414–454.

[Per+14b] Charith Perera, Arkady B. Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. “Sensing as a service model for smart cities supported by Internet of Things”. In: *Transactions on Emerging Telecommunications Technologies* 25.1 (2014), pp. 81–93.

[Per+15] Charith Perera, Rajiv Ranjan, Lizhe Wang, Samee Ullah Khan, and Albert Y. Zomaya. “Big Data Privacy in the Internet of Things Era”. In: *IT Professional* 17.3 (2015), pp. 32–39.

[Per+17a] Charith Perera, Yongrui Qin, Júlio Cezar Estrella, Stephan Reiff-Marganiec, and Athanasios V. Vasilakos. “Fog Computing for Sustainable Smart Cities: A Survey”. In: *ACM Computing Surveys* 50.3 (2017), 32:1–32:43.

[Per+17b] Charith Perera, Susan Wakenshaw, Tim Baarslag, Hamed Haddadi, Arosha K. Bandara, Richard Mortier, Andy Crabtree, Irene Ng, Derek McAuley, and Jon Crowcroft. "Valorising the IoT databox: creating value for everyone". In: *Transactions on Emerging Telecommunications Technologies* 28.1 (2017), pp. 1–17.

[Pie+06] Peter Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh, and Margo Seltzer. "Network-Aware Operator Placement for Stream-Processing Systems". In: *Proc. of the 22nd International Conference on Data Engineering (ICDE)*. 2006, pp. 1–12.

[PL15] Claus Pahl and Brian Lee. "Containers and Clusters for Edge Cloud Architectures - A Technology Review". In: *Proc. of the 3rd International Conference on Future Internet of Things and Cloud*. 2015, pp. 379–386.

[PLM17] Riccardo Petrolo, Valeria Loscrí, and Nathalie Mitton. "Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms". In: *Transactions on Emerging Telecommunications Technologies* 28.1 (2017).

[PM17] Jianli Pan and James McElhannon. "Future edge cloud and edge computing for internet of things applications". In: *IEEE Internet of Things Journal* 5.1 (2017), pp. 439–449.

[Pow+15] Nathaniel Powers, Alexander Alling, Kiara Osolinsky, Tolga Soyata, Meng Zhu, Haoliang Wang, He Ba, Wendi B. Heinzelman, Jiye Shi, and Minseok Kwon. "The Cloudlet Accelerator: Bringing Mobile-Cloud Face Recognition into Real-Time". In: *Proc. of the 2015 IEEE Globecom Workshops*. 2015, pp. 1–7.

[Pre+15] J. S. Preden, K. Tammemäe, A. Jantsch, M. Leier, A. Riid, and E. Calis. "The Benefits of Self-Awareness and Attention in Fog and Mist Computing". In: *Computer* 48.7 (2015), pp. 37–45.

[PS11] B. Pernici and S. H. Siadat. "Selection of Service Adaptation Strategies Based on Fuzzy Logic". In: *Proc. of the 2011 IEEE World Congress on Services*. 2011, pp. 99–106.

[PS18] Jared N. Plumb and Ryan Stutsman. "Exploiting Google's Edge Network for Massively Multiplayer Online Games". In: *Proc. of the 2nd IEEE International Conference on Fog and Edge Computing (ICFEC)*. 2018, pp. 1–8.

[Psa+18] Ioannis Psaras, Onur Ascigil, Sergi Rene, George Pavlou, Alexander Afanasyev, and Lixia Zhang. "Mobile Data Repositories at the Edge". In: *Proc. of the USENIX Workshop on Hot Topics in Edge Computing (HotEdge)*. 2018.

[PSM10] Michael Angelo A. Pedrasa, Ted D. Spooner, and Iain F. MacGill. "Coordinated scheduling of residential distributed energy resources to optimize smart home energy services". In: *IEEE Transactions on Smart Grid* 1.2 (2010), pp. 134–143.

[Pul+18] C. Puliafito, E. Mingozzi, C. Vallati, F. Longo, and G. Merlino. "Virtualization and Migration at the Network Edge: An Overview". In: *Proc. of the 2018 IEEE International Conference on Smart Computing (SMARTCOMP)*. 2018, pp. 368–374.

[Pul+19] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. "Fog Computing for the Internet of Things: A Survey". In: *ACM Transactions on Internet Technology* 19.2 (2019), 18:1–18:41.

[PW02] Lothar Pantel and Lars C. Wolf. "On the Impact of Delay on Real-time Multiplayer Games". In: *Proc. of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV '02. 2002, pp. 23–29.

[PY10] J. T. Piao and J. Yan. "A Network-aware Virtual Machine Placement and Migration Approach in Cloud Computing". In: *Proc. of the 9th International Conference on Grid and Cloud Computing (GCC)*. 2010, pp. 87–92.

[Qiu+18] Hang Qiu, Fawad Ahmad, Fan Bai, Marco Gruteser, and Ramesh Govindan. "AVR: Augmented Vehicular Reality". In: *Proc. of the 16th International Conference on Mobile Systems, Applications, and Services*. MobiSys '18. 2018, pp. 81–95.

[QKB17] Bozhao Qi, Lei Kang, and Suman Banerjee. "A vehicle-based edge computing platform for transit and human mobility analytics". In: *Proc. of the Second ACM/IEEE Symposium on Edge Computing (SEC)*. 2017, 1:1–1:14.

[Ra+11] Moo-Ryong Ra, Anmol Sheth, Lily B. Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. "Odessa: enabling interactive perception applications on mobile devices". In: *Proc. of ACM MobiSys*. 2011, pp. 43–56.

[RAD18] Thomas Rausch, Cosmin Avasalcai, and Schahram Dustdar. "Portable Energy-Aware Cluster-Based Edge Computers". In: *Proc. of the 2018 IEEE/ACM Symposium on Edge Computing (SEC)*. 2018, pp. 260–272.

[RCR11] Meike Ramon, Stéphanie Caharel, and Bruno Rossion. "The speed of recognition of personally familiar faces". In: *Perception* 40 (2011), pp. 437–449.

[RDR10] Stamatia Rizou, Frank Dürr, and Kurt Rothermel. "Solving the multi-operator placement problem in large-scale operator networks". In: *Proc. of the 19th International Conference on Computer Communications (ICCCN)*. 2010, pp. 1–6.

[RDU18] Yefeng Ruan, Arjan Durresi, and Suleyman Uslu. "Trust Assessment for Internet of Things in Multi-access Edge Computing". In: *Proc. of the 32nd IEEE International Conference on Advanced Information Networking and Applications (AINA)*. 2018, pp. 1155–1161.

[Rei+12] Andreas Reinhardt, Paul Baumann, Daniel Burgstahler, Matthias Hollick, Hristo Chonov, Marc Werner, and Ralf Steinmetz. "On the accuracy of appliance identification based on distributed load metering data". In: *Proc. of the 2nd IFIP Conference on Sustainable Internet and ICT for Sustainability (SustainIT)*. 2012, pp. 1–9.

[Ren+18] Yongzheng Ren, Feng Zeng, Wenjia Li, and Lin Meng. "A Low-Cost Edge Server Placement Strategy in Wireless Metropolitan Area Networks". In: *Proc. of the 27th International Conference on Computer Communication and Networks (ICCCN)*. 2018, pp. 1–6.

[Rez+18] Alex Reznik, Anthony Sulisti, Alexander Artemenko, Yonggang Fang, Danny Frydman, Fabio Giust, HuaZhang Lv, Saad Ullah Sheikh, Yifan Yu, and Zhou Zheng. *MEC in an Enterprise Setting: A Solution Outline*. Tech. rep. ETSI, 2018.

[RH05] Maxim Raya and Jean-Pierre Hubaux. “The Security of Vehicular Ad Hoc Networks”. In: *Proc. of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks*. SASN ’05. 2005, pp. 11–21.

[RLM18] Rodrigo Roman, Javier López, and Masahiro Mambo. “Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges”. In: *Future Generation Computer Systems* 78 (2018), pp. 680–698.

[RN16] Flavio Ramalho and Augusto Neto. “Virtualization at the network edge: A performance comparison”. In: *Proc. of the 17th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 2016, pp. 1–6.

[Sal+18] Mohammad A. Salahuddin, Jagruti Sahoo, Roch H. Glitho, Halima Elbiaze, and Wessam Ajib. “A Survey on Content Placement Algorithms for Cloud-Based Content Delivery Networks”. In: *IEEE Access* 6 (2018), pp. 91–114.

[Sam+11] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. “EnerJ: approximate data types for safe and general low-power computation”. In: *Proc. of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 2011, pp. 164–174.

[San+14] Nay Myo Sandar, Sivadon Chaisiri, Sira Yongchareon, and Veronica Liesaputra. “Cloud-Based Video Monitoring Framework: An Approach Based on Software-Defined Networking for Addressing Scalability Problems”. In: *Proc. of Web Information Systems Engineering (WISE) Workshops*. 2014, pp. 176–189.

[Sap+16] Marco Sapienza, Ermanno Guardo, Marco Cavallo, Giuseppe La Torre, Guerrino Leombruno, and Orazio Tomarchio. “Solving Critical Events through Mobile Edge Computing: An Approach for Smart Cities”. In: *Proc. of the 2016 IEEE International Conference on Smart Computing, (SMARTCOMP)*. 2016, pp. 1–5.

[Sat+09] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Cáceres, and Nigel Davies. “The Case for VM-Based Cloudlets in Mobile Computing”. In: *IEEE Pervasive Computing* 8.4 (2009), pp. 14–23.

[Sat+13] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha. “The Role of Cloudlets in Hostile Environments”. In: *IEEE Pervasive Computing* 12.4 (2013), pp. 40–49.

[Sat+14] Mahadev Satyanarayanan, Zhuo Chen, Kiryong Ha, Wenlu Hu, Wolfgang Richter, and Padmanabhan Pillai. “Cloudlets: At the leading edge of mobile-cloud convergence”. In: *Proc. of the 6th International Conference on Mobile Computing, Applications and Services (MobiCASE)*. 2014, pp. 1–9.

[Sat+16] Arjuna Sathiaseelan, Adisorn Lertsinsrubtavee, Adarsh Jagan, Prakash Baskaran, and Jon Crowcroft. “Cloudrone: Micro Clouds in the Sky”. In: *Proc. of the 2nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use (DroNet)*. 2016, pp. 41–44.

[Sat+17] Mahadev Satyanarayanan, Phillip B. Gibbons, Lily B. Mummert, Padmanabhan Pillai, Pieter Simoens, and Rahul Sukthankar. “Cloudlet-based just-in-time indexing of IoT video”. In: *Proc. of the Global Internet of Things Summit (GIoTS 2017)*. 2017, pp. 1–8.

[Sat01] Mahadev Satyanarayanan. “Pervasive computing: vision and challenges”. In: *IEEE Personal Communications* 8.4 (2001), pp. 10–17.

[Sat04] M. Satyanarayanan. “Augmenting Cognition”. In: *IEEE Pervasive Computing* 3.2 (2004), pp. 4–5.

[Sat11] Mahadev Satyanarayanan. “Mobile computing: the next decade”. In: *Mobile Computing and Communications Review* 15.2 (2011), pp. 2–10.

[Sat17] Mahadev Satyanarayanan. “The Emergence of Edge Computing”. In: *IEEE Computer* 50.1 (2017), pp. 30–39.

[SBD18] Meenakshi Syamkumar, Paul Barford, and Ramakrishnan Durairajan. “Deployment Characteristics of "The Edge" in Mobile Edge Computing”. In: *Proc. of the 2018 Workshop on Mobile Edge Communications (MECOMM)*. 2018, pp. 43–49.

[SBH16] Farzad Samie, Lars Bauer, and Jörg Henkel. “IoT Technologies for Embedded Computing: A Survey”. In: *Proc. of the 11th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES ’16. 2016, 8:1–8:10.

[Sce+11] Salvatore Scellato, Cecilia Mascolo, Mirco Musolesi, and Jon Crowcroft. “Track Globally, Deliver Locally: Improving Content Delivery Networks by Tracking Geographic Social Cascades”. In: *Proc. of the 20th International Conference on World Wide Web*. WWW’11. 2011, pp. 457–466.

[Sch+08] E. Schoch, F. Kargl, M. Weber, and T. Leinmüller. “Communication patterns in VANETs”. In: *IEEE Communications Magazine* 46.11 (2008), pp. 119–125.

[Sch+11] Hans Schaffers, Nicos Komninos, Marc Pallot, Brigitte Trousse, Michael Nilsson, and Alvaro Oliveira. “Smart Cities and the Future Internet: Towards Cooperation Frameworks for Open Innovation”. In: *Proc. of the Future Internet Assembly 2011: Achievements and Technological Promises*. 2011, pp. 431–446.

[Sch+12] Immanuel Schweizer, Christian Meurisch, Julien Gedeon, Roman Bärtl, and Max Mühlhäuser. “Noisemap: multi-tier incentive mechanisms for participative urban sensing”. In: *Proc. of the 3rd International Workshop on Sensing Applications on Mobile Phones*. PhoneSense ’12. ACM. 2012, 9:1–9:5.

[Sch+15] Benedikt Schmidt, Sebastian Benchea, Rüdiger Eichin, and Christian Meurisch. “Fitness Tracker or Digital Personal Coach: How to Personalize Training”. In: *Adjunct Proc. of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proc. of the 2015 ACM International Symposium on Wearable Computers*. UbiComp/ISWC’15 Adjunct. 2015, pp. 1063–1067.

[Sch+16a] Dominik Schäfer, Janick Edinger, Justin Mazzola Paluska, Sebastian VanSyckel, and Christian Becker. “Tasklets: "Better than Best-Effort" Computing”. In: *Proc. of the 25th International Conference on Computer Communication and Networks (ICCCN)*. 2016, pp. 1–11.

[Sch+16b] Johannes M. Schleicher, Michael Vögler, Schahram Dustdar, and Christian Inzinger. “Enabling a smart city application ecosystem: Requirements and architectural aspects”. In: *IEEE Internet Computing* 20.2 (2016), pp. 58–65.

[Sch+17] Eve M. Schooler, David Zage, Jeff Sedayao, Hassnaa Moustafa, Andrew Brown, and Moreno Ambrosin. “An Architectural Vision for a Data-Centric IoT: Rethinking Things, Trust and Clouds”. In: *Proc. of the 37th IEEE International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 1717–1728.

[Sco+19] Domenico Scotece, Nafize R. Paiker, Luca Foschini, Paolo Bellavista, Xiaoning Ding, and Cristian Borcea. “MEFS: Mobile Edge File System for Edge-Assisted Mobile Apps”. In: *Proc. of the 20th IEEE International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. 2019, pp. 1–9.

[SF05] Ya-Yunn Su and Jason Flinn. “Slingshot: deploying stateful services in wireless hotspots”. In: *Proc. of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2005, pp. 79–92.

[Sha+11] Bikash Sharma, Victor Chudnovsky, Joseph L. Hellerstein, Rasekh Rifaat, and Chita R. Das. “Modeling and synthesizing task placement constraints in Google compute clusters”. In: *Proc. of the ACM Symposium on Cloud Computing (SOCC)*. 2011, pp. 1–14.

[Shi+11] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. “Privacy-Preserving Aggregation of Time-Series Data”. In: *Proc. of the Network and Distributed System Security Symposium (NDSS)*. 2011, pp. 1–17.

[Shi+13] Muhammad Shiraz, Saeid Abolfazli, Zohreh Sanaei, and Abdullah Gani. “A study on virtual machine deployment for application outsourcing in mobile cloud computing”. In: *The Journal of Supercomputing* 63.3 (2013), pp. 946–964.

[Shi+14] Cong Shi, Karim Habak, Pranesh Pandurangan, Mostafa Ammar, Mayur Naik, and Ellen Zegura. “COSMOS: Computation Offloading as a Service for Mobile Devices”. In: *Proc. of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. MobiHoc ’14. 2014, pp. 287–296.

[Shi+15] Jinghao Shi, Liwen Gui, Dimitrios Koutsonikolas, Chunming Qiao, and Geoffrey Challen. “A Little Sharing Goes a Long Way: The Case for Reciprocal Wifi Sharing”. In: *Proc. of the 2nd International Workshop on Hot Topics in Wireless*. HotWireless ’15. 2015, pp. 6–10.

[Shi+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. “Edge Computing: Vision and Challenges”. In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646.

[Shi+19] Shu Shi, Varun Gupta, Michael Hwang, and Rittwik Jana. “Mobile VR on Edge Cloud: A Latency-driven Design”. In: *Proc. of the 10th ACM Multimedia Systems Conference*. MMSys ’19. 2019, pp. 222–231.

[Sid+11] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin C. Rinard. “Managing performance vs. accuracy trade-offs with loop perforation”. In: *Proc. of the SIGSOFT/FSE’11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC’11: 13th European Software Engineering Conference (ESEC-13)*. 2011, pp. 124–134.

[Sil+16] Ana Cristina Franco da Silva, Uwe Breitenbücher, Kálmán Képes, Oliver Kopp, and Frank Leymann. “OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker”. In: *Proc. of the 6th International Conference on the Internet of Things (IOT)*. 2016, pp. 181–182.

[Sil16] Alan Sill. “The Design and Architecture of Microservices”. In: *IEEE Cloud Computing* 3.5 (2016), pp. 76–80.

[Sim+13] Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, and Mahadev Satyanarayanan. “Scalable crowd-sourcing of video from mobile devices”. In: *Proc. of the 11th International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2013, pp. 139–152.

[SKK12] M. Sharifi, S. Kafaie, and O. Kashefi. “A Survey and Taxonomy of Cyber Foraging of Mobile Devices”. In: *IEEE Communications Surveys Tutorials* 14.4 (2012), pp. 1232–1243.

[SLM17] L. Sun, Y. Li, and R. A. Memon. “An open IoT framework based on microservices architecture”. In: *China Communications* 14.2 (2017), pp. 154–162.

[SMR13] Luis Emiliano Sanchez, Sabine Moisan, and Jean-Paul Rigault. “Metrics on feature models to optimize configuration adaptation at run time”. In: *Proc. of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering, (CMSBSE)*. 2013, pp. 39–44.

[SMT10] Patrick Stuedi, Iqbal Mohomed, and Doug Terry. “WhereStore: Location-based Data Storage for Mobile Devices Interacting with the Cloud”. In: *Proc. of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. MCS ’10. 2010, 1:1–1:8.

[SP15] Fabrice Starks and Thomas Peter Plagemann. “Operator placement for efficient distributed complex event processing in MANETs”. In: *Proc. of the 11th IEEE International Conference on Wireless and Mobile Computing (WiMob)*. 2015, pp. 83–90.

[Spi17] Josef Spillner. “Snafu: Function-as-a-Service (FaaS) Runtime Design and Implementation”. In: *CoRR* abs/1703.07562 (2017), pp. 1–15.

[SS13] O. Saleh and K. Sattler. “Distributed Complex Event Processing in Sensor Networks”. In: *Proc. of the 2013 IEEE 14th International Conference on Mobile Data Management*. 2013, pp. 23–26.

[SS14] Immanuel Schweizer and Benedikt Schmidt. “Kraken.Me: Multi-device User Tracking Suite”. In: *Proc. of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. UbiComp ’14 Adjunct. 2014, pp. 853–862.

[Ste92] Jonathan Steuer. “Defining Virtual Reality: Dimensions Determining Telepresence”. In: *Journal of Communication* 42.4 (1992), pp. 73–93.

[Ste97] Robert Stephens. “A Survey of Stream Processing”. In: *Acta Informatica* 34.7 (1997), pp. 491–541.

[STH18] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan van den Heuvel. “The pains and gains of microservices: A Systematic grey literature review”. In: *Journal of Systems and Software* 146 (2018), pp. 215–232.

[Sto+16] Ivan Stojmenovic, Sheng Wen, Xinyi Huang, and Hao Luan. “An overview of Fog computing and its security issues”. In: *Concurrency and Computation: Practice and Experience* 28.10 (2016), pp. 2991–3005.

[Sto12] Milos Stojmenovic. “Mobile Cloud Computing for Biometric Applications”. In: *Proc. of the 15th International Conference on Network-Based Information Systems (NBiS 2012)*. 2012, pp. 654–659.

[Swa12] Melanie Swan. “Health 2050: The Realization of Personalized Medicine through Crowdsourcing, the Quantified Self, and the Participatory Biocitizen”. In: *Journal of Personalized Medicine* 2.3 (2012), pp. 93–118.

[Tal+17] Tarik Taleb, Sunny Dutta, Adlen Ksentini, Muddesar Iqbal, and Hannu Flinck. “Mobile Edge Computing Potential in Making Cities Smarter”. In: *IEEE Communications Magazine* 55.3 (2017), pp. 38–43.

[Tan+15] Haowen Tang, Fangming Liu, Guobin Shen, Yuchen Jin, and Chuanxiong Guo. “UniDrive: Synergize Multiple Consumer Cloud Storage Services”. In: *Proc. of the 16th Annual Middleware Conference*. Middleware ’15. ACM, 2015, pp. 137–148.

[Tha+18] Jörg Thalheim, Pramod Bhatotia, Pedro Fonseca, and Baris Kasikci. “CNTR: Lightweight OS Containers”. In: *Proc. of the 2018 USENIX Annual Technical Conference (ATC)*. 2018, pp. 199–212.

[Tho12] Bruce H. Thomas. “A Survey of Visual, Mixed, and Augmented Reality Gaming”. In: *Computers in Entertainment* 10.1 (2012), 3:1–3:33.

[TLL14] Cory Thoma, Alexandros Labrinidis, and Adam J. Lee. “Automated operator placement in distributed Data Stream Management Systems subject to user constraints”. In: *Workshops Proc. of the 30th International Conference on Data sEngineering Workshops (ICDE Workshops)*. 2014, pp. 310–316.

[TLP17] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. “Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation”. In: *IEEE Cloud Computing* 4.5 (2017), pp. 22–32.

[Tra+17] T. X. Tran, P. Pandey, A. Hajisami, and D. Pompili. “Collaborative multi-bitrate video caching and processing in Mobile-Edge Computing networks”. In: *Proc. of the 2017 13th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. 2017, pp. 165–172.

[Tri+19] Marco Trinelli, Massimo Gallo, Myriana Rifai, and Fabio Pianese. “Transparent AR Processing Acceleration at the Edge”. In: *Proc. of the 2nd International Workshop on Edge Systems, Analytics and Networking (EdgeSys)*. 2019, pp. 30–35.

[Tsa+17] P. Tsai, H. Hong, A. Cheng, and C. Hsu. “Distributed analytics in fog computing platforms using tensorflow and kubernetes”. In: *Proc. of the 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. 2017, pp. 145–150.

[TVM18] G. Tanganelli, C. Vallati, and E. Mingozzi. “Edge-Centric Distributed Discovery and Access in the Internet of Things”. In: *IEEE Internet of Things Journal* 5.1 (2018), pp. 425–438.

[Val+16] Carlo Vallati, Antonio Virdis, Enzo Mingozzi, and Giovanni Stea. “Mobile-edge computing come home connecting things in future smart homes using LTE device-to-device communications”. In: *IEEE Consumer Electronics Magazine* 5.4 (2016), pp. 77–83.

[Var+16] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos. “Challenges and Opportunities in Edge Computing”. In: *Proc. of the 2016 IEEE International Conference on Smart Cloud (SmartCloud)*. 2016, pp. 20–26.

[Var+17] Blesson Varghese, Nan Wang, Dimitrios S. Nikolopoulos, and Rajkumar Buyya. “Feasibility of Fog Computing”. In: *CoRR* abs/1701.05451 (2017), pp. 1–8.

[Vas+15] Emmanouil Vasilomanolakis, Jörg Daubert, Manisha Luthra, Vangelis Gazis, Alexander Wiesmaier, and Panayotis Kikiras. “On the Security and Privacy of Internet of Things Architectures and Systems”. In: *Proc. of the 2015 International Workshop on Secure Internet of Things (SIoT)*. 2015, pp. 49–57.

[vBS01] J. van Gurp, J. Bosch, and M. Svahnberg. “On the notion of variability in software product lines”. In: *Proc. of the Working IEEE/IFIP Conference on Software Architecture*. 2001, pp. 45–54.

[Ver+12a] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. "Cloudlets: Bringing the Cloud to the Mobile User". In: *Proc. of the 3rd ACM Workshop on Mobile Cloud Computing and Services (MCS)*. 2012, pp. 29–36.

[Ver+12b] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. "AIOLOS: Middleware for improving mobile application performance through cyber foraging". In: *Journal of Systems and Software* 85.11 (2012), pp. 2629–2639.

[Vii+18] M. Viitanen, J. Vanne, T. D. Hämäläinen, and A. Kulmala. "Low Latency Edge Rendering Scheme for Interactive 360 Degree Virtual Reality Gaming". In: *Proc. of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 1557–1560.

[Vil+15a] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud". In: *Proc.of the 2015 10th Computing Colombian Conference (10CCC)*. 2015, pp. 583–590.

[Vil+15b] Felix Jesús Villanueva, David Villa, Maria J. Santofimia, Jesús Barba, and Juan Carlos López. "Crowdsensing smart city parking monitoring". In: *Proc. of the 2nd IEEE World Forum on Internet of Things (WF-IoT)*. 2015, pp. 751–756.

[Vil+16a] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang. "Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures". In: *Proc. of the 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 2016, pp. 179–182.

[Vil+16b] Massimo Villari, Maria Fazio, Schahram Dustdar, Omer F. Rana, and Rajiv Ranjan. "Osmotic Computing: A New Paradigm for Edge/Cloud Integration". In: *IEEE Cloud Computing* 3.6 (2016), pp. 76–83.

[VP03] Athena Vakali and George Pallis. "Content Delivery Networks: Status and Trends". In: *IEEE Internet Computing* 7.6 (2003), pp. 68–74.

[VR14] Luis M. Vaquero and Luis Rodero-Merino. "Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing". In: *SIGCOMM Computer Communication Review* 44.5 (2014), pp. 27–32.

[Vul+15] Ashish Vulimiri, Carlo Curino, P. Brighten Godfrey, Thomas Jungblut, Jitu Padhye, and George Varghese. "Global Analytics in the Face of Bandwidth and Regulatory Constraints". In: *Proc. of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2015, pp. 323–336.

[Wag19] Martin Wagner. "A Microservice-Based Offloading Architecture for Edge Computing". Master's Thesis. Technische Universität Darmstadt, Department of Computer Science, 2019.

[Wan+12] Zhaoran Wang, Yu Zhang, Xiaotao Chang, Xiang Mi, Yu Wang, Kun Wang, and Huazhong Yang. “Pub/Sub on stream: a multi-core based message broker with QoS support”. In: *Proc. of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS)*. 2012, pp. 127–138.

[Wan+15] Lin Wang, Antonio Fernández Anta, Fa Zhang, Jie Wu, and Zhiyong Liu. “Multi-resource energy-efficient routing in cloud data centers with network-as-a-service”. In: *Proc. of the 2015 IEEE Symposium on Computers and Communication (ISCC)*. 2015, pp. 694–699.

[Wan+16] C. Wang, Y. Li, D. Jin, and S. Chen. “On the Serviceability of Mobile Vehicular Cloudlets in a Large-Scale Urban Environment”. In: *IEEE Transactions on Intelligent Transportation Systems* 17.10 (2016), pp. 2960–2970.

[Wan+17] Qixu Wang, Dajiang Chen, Ning Zhang, Zhe Ding, and Zhiguang Qin. “PCP: A Privacy-Preserving Content-Based Publish-Subscribe Scheme With Differential Privacy in Fog Computing”. In: *IEEE Access* 5 (2017), pp. 17962–17974.

[Wan+18a] Jiafu Wan, Baotong Chen, Shiyong Wang, Min Xia, Di Li, and Chengliang Liu. “Fog computing for energy-aware load balancing and scheduling in smart factory”. In: *Transactions on Industrial Informatics* 14.10 (2018), pp. 4548–4556.

[Wan+18b] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S. Yang, and M. Satyanarayanan. “Bandwidth-Efficient Live Video Analytics for Drones Via Edge Computing”. In: *Proc. of the 2018 IEEE/ACM Symposium on Edge Computing (SEC)*. 2018, pp. 159–173.

[Wan+18c] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael M. Swift. “Peeking Behind the Curtains of Serverless Platforms”. In: *Proc. of the 2018 USENIX Annual Technical Conference (ATC)*. 2018, pp. 133–146.

[Wan+19] Lin Wang, Lei Jiao, Jun Li, Julien Gedeon, and Max Mühlhäuser. “MOERA: Mobility-agnostic Online Resource Allocation for Edge Computing”. In: *IEEE Transactions on Mobile Computing* 18.8 (2019), pp. 1843–1856.

[Wan06] Roy Want. “An Introduction to RFID Technology”. In: *IEEE Pervasive Computing* 5.1 (2006), pp. 25–33.

[Wan11] Bang Wang. “Coverage Problems in Sensor Networks: A Survey”. In: *ACM Computing Surveys* 43.4 (2011), 32:1–32:53.

[WD10] Shaoxuan Wang and Sujit Dey. “Rendering Adaptation to Address Communication and Computation Constraints in Cloud Mobile Gaming”. In: *Proc. of the Global Communications Conference (GLOBECOM)*. 2010, pp. 1–6.

[Wec+18] Markus Weckesser, Roland Kluge, Martin Pfannemüller, Michael Matthé, Andy Schürr, and Christian Becker. “Optimal reconfiguration of dynamic software product lines based on performance-influence models”. In: *Proc. of the 22nd International Systems and Software Product Line Conference (SPLC)*. 2018, pp. 98–109.

[Wen+18] Zhenyu Wen, Do Le Quoc, Pramod Bhatotia, Ruichuan Chen, and Myungjin Lee. “ApproxIoT: Approximate Analytics for Edge Computing”. In: *Proc. of the 38th IEEE International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 411–421.

[Wol+17] Sjaak Wolfert, Lan Ge, Cor Verdouw, and Marc-Jeroen Bogaardt. “Big Data in Smart Farming – A review”. In: *Agricultural Systems* 153 (2017), pp. 69–80.

[Wu+13] Hsin-Kai Wu, Silvia Wen-Yu Lee, Hsin-Yi Chang, and Jyh-Chong Liang. “Current status, opportunities and challenges of augmented reality in education”. In: *Computers & Education* 62 (2013), pp. 41–49.

[Wu+17a] Dazhong Wu, Shaopeng Liu, Li Zhang, Janis Terpenny, Robert X. Gao, Thomas Kurfess, and Judith A. Guzzo. “A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing”. In: *Journal of Manufacturing Systems* 43 (2017), pp. 25–34.

[Wu+17b] Song Wu, Chao Niu, Jia Rao, Hai Jin, and Xiaohai Dai. “Container-Based Cloud Platform for Mobile Computation Offloading”. In: *Proc. of the 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2017, pp. 123–132.

[Wu+18] Song Wu, Chao Mei, Hai Jin, and Duoqiang Wang. “Android Unikernel: Gearing mobile code offloading towards edge computing”. In: *Future Generation Computer Systems* 86 (2018), pp. 694–703.

[WUS15] Y. Wang, T. Uehara, and R. Sasaki. “Fog Computing: Issues and Challenges in Security and Forensics”. In: *Proc. of the 2015 IEEE 39th Annual Computer Software and Applications Conference*. 2015, pp. 53–59.

[XGR18] Zhuangdi Xu, Harshit Gupta, and Umakishore Ramachandran. “STTR: A System for Tracking All Vehicles All the Time At the Edge of the Network”. In: *Proc. of the 12th ACM International Conference on Distributed and Event-based Systems*. DEBS ’18. 2018, pp. 124–135.

[Xio+18] Ying Xiong, Yulin Sun, Li Xing, and Ying Huang. “Extend Cloud to Edge with KubeEdge”. In: *Proc. of the IEEE/ACM Symposium on Edge Computing (SEC)*. 2018, pp. 373–377.

[XK17] Yong Xiao and Marwan Krunz. “QoE and power efficiency tradeoff for fog computing networks with fog node cooperation”. In: *Proc. of the IEEE Conference on Computer Communications (INFOCOM)*. 2017, pp. 1–9.

[XMK16] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. “Approximate Computing: A Survey”. In: *IEEE Design & Test* 33.1 (2016), pp. 8–22.

[Xu+14] Song Xu, Manuela Pérez, Kun Yang, Cyril Perrenot, Jacques Felblinger, and Jacques Hubert. “Determination of the latency effects on surgical performance and the acceptable latency levels in telesurgery using the dV-Trainer® simulator”. In: *Surgical endoscopy* 28.9 (2014), pp. 2569–2576.

[Xu+15] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo. “Capacitated cloudlet placements in Wireless Metropolitan Area Networks”. In: *Proc. of the 40th IEEE Conference on Local Computer Networks (LCN)*. 2015, pp. 570–578.

[Xu+16] Zichuan Xu, Weifa Liang, Wenzheng Xu, Mike Jia, and Song Guo. “Efficient Algorithms for Capacitated Cloudlet Placements”. In: *IEEE Transactions on Parallel and Distributed Systems* 27.10 (2016), pp. 2866–2880.

[Yan+10] Zhi Yang, Ben Y. Zhao, Yuanjian Xing, Song Ding, Feng Xiao, and Yafei Dai. “AmazingStore: available, low-cost online storage service using cloudlets”. In: *Proc. of the 9th international conference on Peer-to-peer systems (IPTPS)*. 2010, pp. 1–5.

[Yan+16] G. Yang, Q. Sun, A. Zhou, S. Wang, and J. Li. “Poster Abstract: Access Point Ranking for Cloudlet Placement in Edge Computing Environment”. In: *Proc. of the 2016 IEEE/ACM Symposium on Edge Computing (SEC)*. 2016, pp. 85–86.

[Yao+17] Hong Yao, Changmin Bai, Muzhou Xiong, Deze Zeng, and Zhangjie Fu. “Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing”. In: *Concurrency and Computation: Practice and Experience* 29.16 (2017), pp. 1–9.

[Ye+13] Rong Ye, Ting Wang, Feng Yuan, Rakesh Kumar, and Qiang Xu. “On reconfiguration-oriented approximate adder design and its application”. In: *Proc. of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2013, pp. 48–54.

[Yi+15] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. “Fog Computing: Platform and Applications”. In: *Proc. of the Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb 2015)*. 2015, pp. 73–78.

[Yi+17] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li. “LAVEA: Latency-Aware Video Analytics on Edge Computing Platform”. In: *Proc. of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 2573–2574.

[Yin+17] Hao Yin, Xu Zhang, Hongqiang Harry Liu, Yan Luo, Chen Tian, Shuoyao Zhao, and Feng Li. “Edge Provisioning with Flexible Server Placement”. In: *IEEE Transactions on Parallel and Distributed Systems* 28.4 (2017), pp. 1031–1045.

[YLL15] Shanhe Yi, Cheng Li, and Qun Li. “A Survey of Fog Computing: Concepts, Applications and Issues”. In: *Proc. of the 2015 Workshop on Mobile Big Data (Mobidata)*. 2015, pp. 37–42.

[YLN03] Jungkeun Yoon, Mingyan Liu, and Brian Noble. “Sound Mobility Models”. In: *Proc. of the 9th Annual International Conference on Mobile Computing and Networking (MOBICOM)*. Ed. by David B. Johnson, Anthony D. Joseph, and Nitin H. Vaidya. 2003, pp. 205–216.

[You+17] C. You, K. Huang, H. Chae, and B. Kim. “Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading”. In: *IEEE Transactions on Wireless Communications* 16.3 (2017), pp. 1397–1411.

[You+19] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. “All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey”. In: *Journal of Systems Architecture* 98 (2019), pp. 289–330.

[YQL15] Shanhe Yi, Zhengrui Qin, and Qun Li. “Security and Privacy Issues of Fog Computing: A Survey”. In: *Proc. of the 10th International Conference on Wireless Algorithms, Systems, and Applications (WASA)*. 2015, pp. 685–695.

[Yu+18] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang. “A Survey on the Edge Computing for the Internet of Things”. In: *IEEE Access* 6 (2018), pp. 6900–6919.

[Yua+18] Q. Yuan, H. Zhou, J. Li, Z. Liu, F. Yang, and X. S. Shen. “Toward Efficient Content Delivery for Automated Driving Services: An Edge Computing Solution”. In: *IEEE Network* 32.1 (2018), pp. 80–86.

[ZA08] Qian Zhu and Gagan Agrawal. “Resource Allocation for Distributed Streaming Applications”. In: *Proc. of the 2008 International Conference on Parallel Processing (ICPP)*. 2008, pp. 414–421.

[Zam+17] Ali Reza Zamani, Ioan Petri, Javier Diaz Montes, Omer F. Rana, and Manish Parashar. “Edge-Supported Approximate Analysis for Long Running Computations”. In: *Proc. of the 5th IEEE International Conference on Future Internet of Things and Cloud (FiCloud)*. 2017, pp. 321–328.

[Zan+14] Andrea Zanella, Nicola Bui, Angelo Paolo Castellani, Lorenzo Vangelista, and Michele Zorzi. “Internet of Things for Smart Cities”. In: *IEEE Internet of Things Journal* 1.1 (2014), pp. 22–32.

[Zao+14] John K. Zao, Tchin Tze Gan, Chun Kai You, Sergio Jose Rodriguez Mendez, Cheng En Chung, Yu-Te Wang, Tim R. Mullen, and Tzyy-Ping Jung. “Augmented Brain Computer Interaction Based on Fog Computing and Linked Data”. In: *Proc. of the 2014 International Conference on Intelligent Environments*. 2014, pp. 374–377.

[Zen+18] M. Zeng, Y. Li, K. Zhang, M. Waqas, and D. Jin. “Incentive Mechanism Design for Computation Offloading in Heterogeneous Fog Computing: A Contract-Based Approach”. In: *Proc. of the 2018 IEEE International Conference on Communications (ICC)*. 2018, pp. 1–6.

[Zey+16] Engin Zeydan, Ejder Bastug, Mehdi Bennis, Manhal Abdel Kader, Ilyas Alper Karatepe, Ahmet Salih Er, and Mérouane Debbah. “Big data caching for networking: moving from cloud to edge”. In: *IEEE Communications Magazine* 54.9 (2016), pp. 36–42.

[Zha+14] Qian Zhang, Feng Yuan, Rong Ye, and Qiang Xu. “ApproxIt: An Approximate Computing Framework for Iterative Methods”. In: *Proc. of the 51st Annual Design Automation Conference 2014 (DAC)*. 2014, 97:1–97:6.

[Zha+15a] Feixiong Zhang, Chenren Xu, Yanyong Zhang, K. K. Ramakrishnan, Shreyasee Mukherjee, Roy D. Yates, and Thu D. Nguyen. "EdgeBuffer: Caching and prefetching content at the edge in the MobilityFirst future Internet architecture". In: *Proc. of the 16th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 2015, pp. 1–9.

[Zha+15b] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. "ApproxANN: an approximate computing framework for artificial neural network". In: *Proc. of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2015, pp. 701–706.

[Zha+17] Wuyang Zhang, Jiachen Chen, Yanyong Zhang, and Dipankar Raychaudhuri. "Towards efficient edge cloud augmentation for virtual reality MMOGs". In: *Proc. of the Second ACM/IEEE Symposium on Edge Computing (SEC)*. 2017, 8:1–8:14.

[Zha+18a] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A. Lee. "AWStream: adaptive wide-area streaming analytics". In: *Proc. of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 2018, pp. 236–252.

[Zha+18b] D. Zhang, Y. Ma, Y. Zhang, S. Lin, X. S. Hu, and D. Wang. "A Real-Time and Non-Cooperative Task Allocation Framework for Social Sensing Applications in Edge Computing Systems". In: *Proc. of the 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2018, pp. 316–326.

[Zha+18c] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu. "Data Security and Privacy-Preserving in Edge Computing Paradigm: Survey and Open Issues". In: *IEEE Access* 6 (2018), pp. 18209–18237.

[Zha+18d] J. Zhang, X. Hu, Z. Ning, E.C.H. -. Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu. "Energy-Latency Tradeoff for Energy-Aware Offloading in Mobile Edge Computing Networks". In: *IEEE Internet of Things Journal* 5.4 (2018), pp. 2633–2645.

[Zha+18e] Lei Zhao, Wen Sun, Yongpeng Shi, and Jiajia Liu. "Optimal Placement of Cloudlets for Access Delay Minimization in SDN-Based Internet of Things Networks". In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 1334–1344.

[Zhe+11] Yu Zheng, Yanchi Liu, Jing Yuan, and Xing Xie. "Urban Computing with Taxicabs". In: *Proc. of the 13th International Conference on Ubiquitous Computing*. UbiComp '11. 2011, pp. 89–98.

[Zhe+14a] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. "Urban Computing: Concepts, Methodologies, and Applications". In: *ACM Transactions on Intelligent Systems and Technology* 5.3 (2014), 38:1–38:55.

[Zhe+14b] Yu Zheng, Tong Liu, Yilun Wang, Yanmin Zhu, Yanchi Liu, and Eric Chang. "Diagnosing New York City's Noises with Ubiquitous Data". In: *Proc. of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '14. 2014, pp. 715–725.

[Zho+15] S. Zhou, K. Lin, J. Na, C. Chuang, and C. Shih. “Supporting Service Adaptation in Fault Tolerant Internet of Things”. In: *Proc. of the 2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA)*. 2015, pp. 65–72.

[Zho+17] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya. “mCloud: A Context-Aware Offloading Framework for Heterogeneous Mobile Cloud”. In: *IEEE Transactions on Services Computing* 10.5 (2017), pp. 797–810.

[Zhu+13] J. Zhu, D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu, and F. Bonomi. “Improving Web Sites Performance Using Edge Servers in Fog Computing Architecture”. In: *Proc. of the 2013 IEEE 7th International Symposium on Service-Oriented System Engineering*. 2013, pp. 320–323.

[Zhu+15] Wen-Yuan Zhu, Wen-Chih Peng, Ling-Jyh Chen, Kai Zheng, and Xiaofang Zhou. “Modeling User Mobility for Location Promotion in Location-based Social Networks”. In: *Proc. of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2015, pp. 1573–1582.

[ZLH13] Yu Zheng, Furui Liu, and Hsun-Ping Hsieh. “U-Air: When Urban Air Quality Inference Meets Big Data”. In: *Proc. of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’13. 2013, pp. 1436–1444.

[ZMN05] F. Zhu, M. W. Mutka, and L. M. Ni. “Service discovery in pervasive computing environments”. In: *IEEE Pervasive Computing* 4.4 (2005), pp. 81–90.

Appendices

APPENDIX A

Access Point Location Estimation from Wardriving

LISTING A.1: ACCESSPOINT.RB SOURCE CODE FILE

```ruby

require_relative 'utils'
require 'geoutm'

class AccessPoint
  attr_accessor :bssid
  attr_accessor :ssid
  attr_accessor :measurement
  attr_accessor :caps
  attr_accessor :vendor

  def initialize(bssid, ssid, vendor, frequency, caps, security, timestamp,lat,long,signalstrength,distance)
    @measurement = []
    coordinate = GeoUtm::LatLon.new(lat.to_f, long.to_f)
    utm = coordinate.to_utm()
    @measurement<<[timestamp,utm.e,utm.n,signalstrength,distance]
    @bssid = bssid
    @ssid = ssid
    @vendor = vendor
    @frequency = frequency
    @caps = caps
    @securtiy = security
    @iter = 2000;
    @alpha = 2.0;
    @ratio = 0.99;
    @earthR = 6371
  end

  def addMeasurement(timestamp,lat,long,signalstrength,distance)
    utm = GeoUtm::LatLon.new(lat.to_f, long.to_f).to_utm()
    @measurement<<[timestamp,utm.e,utm.n,signalstrength,distance]
  end

  def dist(x_1,x_2,y_1,y_2)
    return Math.sqrt( (x_1.to_f - x_2.to_f)**2 + (y_1.to_f - y_2.to_f)**2 )
  end

  def estimatePosition
    numOfMeasurementPoints = @measurement.size
    if numOfMeasurementPoints < 3
      return false
    end
    lats = []
    longs = []
    @measurement.each do |mm|
      gps = GeoUtm::UTM.new('32U', mm[1], mm[2], ellipsoid = GeoUtm::Ellipsoid::WGS84)
      latlong = gps.to_lat_lon
      lats << latlong.lat.to_f
      longs << latlong.lon.to_f
    end
    avglat = lats.inject {|sum,lat| sum + lat} / lats.size
    avglong = longs.inject{|sum,long| sum + long} / longs.size

    newmm = []
    @measurement.each do |mm|
      gps = GeoUtm::UTM.new('32U', mm[1], mm[2], ellipsoid = GeoUtm::Ellipsoid::WGS84)
      latlong = gps.to_lat_lon
      lat = latlong.lat
      long = latlong.lon
      if distance(lat,long, avglat, avglong) > 10
        newmm << mm
      end
    end

    @measurement = newmm
    numOfMeasurementPoints = @measurement.size
    return nil if numOfMeasurementPoints == 0

    delta = [0.0,0.0]
    alpha = @alpha
    utm_start = GeoUtm::UTM.new('32U',@measurement[0][1], @measurement[0][2])
    res = [0.0,0.0]
    for iter in 0..@iter
      delta = [0.0,0.0]
      @measurement.each do |mm|
        d = dist(res[0],mm[1].to_f,res[1],mm[2].to_f)
        diff = [((mm[1].to_f-res[0]) *
            (alpha * (d-mm[4].to_f) / [mm[4].to_f,d].max)),
            ((mm[2].to_f-res[1]) *
            (alpha * (d-mm[4].to_f) / [mm[4].to_f,d].max))]
        delta[0] = delta[0] + diff[0]
        delta[1] = delta[1] + diff[1]
      end
      delta[0] = delta[0] * (1.0 / @measurement.length)
      delta[1] = delta[1] * (1.0 / @measurement.length)
      alpha = alpha * @ratio
      res[0] = res[0] + delta[0];
      res[1] = res[1] + delta[1];
    end
    utm_coordinate = GeoUtm::UTM.new('32U', res[0], res[1], ellipsoid = GeoUtm::Ellipsoid::WGS84)
    latlong = utm_coordinate.to_lat_lon
    return latlong.lat.to_s, latlong.lon.to_s
  end
end
```

APPENDIX B

TOSCA Extension for the Description of Microservices[1]

```
##################################################################
# The content of this file reflects TOSCA microservices Profile in
# YAML version 1.0.0. It describes the definition for TOSCA
# microservice types including Node Type, Relationship Type,
# Capability Type and Interfaces.
##################################################################
tosca_definitions_version:
    tosca_simple_profile_for_microservices_1_0_0

##################################################################
# Node Type.
# A Node Type is a reusable entity that defines the type of one or
# more Node Templates.
##################################################################
node_types:
  tosca.nodes.microservices:
    derived_from: tosca.nodes.Root
    description: Base type for microservice definitions.
    properties:
      id:
        type: string
        description: ID of this microservice
      name:
        type: string
        description: Name of this microservice
      mem_requirement:
        description: Required memory in MB for this microservice.
        type: integer
      inputs:
```

[1] **Contribution statement:** I led the idea generation and design of the flexEdge framework. The framework itself was implemented by Martin Wagner as part of this Master thesis [Wag19]. The TOSCA extension shown here is taken from this implementation.

```
      description: a list of inputs this microservice accepts
      type: tosca.datatypes.microservice_inputs
      required: false
    outputs:
      description: a list of outputs this microservice accepts
      type: tosca.datatypes.microservice_outputs
      required: false
    category:
      description: the category path to which this microservice
          belongs to
      type: string
      required: true
    alive_time:
      description: the time in seconds after which this
          microservice will be stopped
      type: integer
      required: false
      default: 300
    wait_for_start:
      description: defines if an agent should wait for a start
          message sent by the microservice
      type: boolean
      required: false
      default: false
    multiple_user_support:
      description: defines if the microservice supports multiple
          users
      type: boolean
      required: false
      default: false

tosca.nodes.microservices.docker_container:
  derived_from: tosca.nodes.microservices
  description: Base type for docker container microservice
      definitions.
  properties:
    bridge_network:
      description: Name of the user-defined bridge network, to
          which this microservice should connect
      type: string
      required: false
    container_port:
      description: Port of the container/application
      type: tosca.datatypes.network.PortDef
      required: false
    container_ports:
      description: List of ports of the container/application
      type: tosca.datatypes.microservice_port_list
      required: false
    host_port:
      description: Port which will be exposed on the host, a
          random port will get exposed if it is not defined
      type: tosca.datatypes.network.PortDef
      required: false
      host_ports:
        description: List of ports which will be exposed on the
            host, random ports will get exposed if it is not
            defined
        type: tosca.datatypes.microservice_port_list
        required: false
      directory:
        description: Name of the directory, in which the dockerfile
             is located
        type: string
        required: true

  tosca.nodes.microservices.unikernel:
    derived_from: tosca.nodes.microservices
    description: Base type for unikernel microservice definitions.

##################################################################
# Relationship Type.
# A Relationship Type is a reusable entity that defines the type
    of
# one or more relationships between Node Types or Node Templates.
##################################################################
relationship_types:
  tosca.relationships.docker_bridge_network:
    derived_from: tosca.relationships.Root
    description: Relationship to set up a docker user-defined
        bridge network
    properties:
      name:
        type: string
        description: Name of this bridge network

##################################################################
# Data Type.
# A Datatype is a complex data type declaration which contains
# other complex or simple data types.
##################################################################
data_types:
  tosca.datatypes.microservice_io:
    derived_from: tosca.datatypes.Root
    type: string
    description: a datatype for defining the type of an input or
        output
    constraints:
      - valid_values: [ number, integer, float, string, text_file,
         image, list ]

  tosca.datatypes.microservice_inputs:
    derived_from: tosca.datatypes.Root
    type: list
    description: a list of constrained strings to define the inputs
         of a microservice
    entry_schema:
      type: tosca.datatypes.microservice_io
  tosca.datatypes.microservice_outputs:
    derived_from: tosca.datatypes.Root
    type: list
    description: a list of constrained strings to define the
        outputs of a microservice
    entry_schema:
      type: tosca.datatypes.microservice_io

  tosca.datatypes.microservice_port:
    derived_from: tosca.datatypes.Root
    description: description for a port, consisting of the port
        number, protocol, and a description string
    properties:
      port:
        type: tosca.datatypes.network.PortDef
        required: true
      protocol:
        type: string
        required: true
        default: tcp
        constraints:
          - valid_values: [ tcp, udp ]
      description:
        type: string
        required: false

  tosca.datatypes.microservice_port_list:
    derived_from: tosca.datatypes.Root
    description: a list of port descriptions to define multiple
        ports
    type: list
    entry_schema:
      type: tosca.datatypes.microservice_port

##################################################################
# Group Type.
# Group Type represents logical grouping of TOSCA nodes that have
# an implied membership relationship and may need to be
# orchestrated or managed together to achieve some result.
##################################################################
group_types:
  tosca.groups.docker_bridge_network:
    derived_from: tosca.groups.Root
    description: Group type to set up a docker user—defined bridge
        network
    properties:
      name:
        type: string
        description: Name of this bridge network
```

APPENDIX C

TOSCA Extension for the Description of Service Chains[1]

```
##################################################################
# The content of this file reflects TOSCA microservices Profile in
# YAML version 1.0.0. It describes the definition for TOSCA
# microservice chain types including Node Type, Relationship Type,
# Capability Type and Interfaces.
##################################################################
tosca_definitions_version:
    tosca_simple_profile_for_microservice_chains_1_0_0

##################################################################
# Node Type.
# A Node Type is a reusable entity that defines the type of one or
# more Node Templates.
##################################################################
node_types:
    tosca.nodes.chained_microservice:
        derived_from: tosca.nodes.Root
        description: base type for chained microservices
        properties:
            store_id:
                type: string
                description: store ID of microservice to use, has
                    higher priority than semantic description
                required: false
            semantic_description:
                type: tosca.datatypes.semantic_description
                description: semantic description of microservice
                    to use, has lower priority than ID
                required: false
```

[1] **Contribution statement:** I led the idea generation and design of the flexEdge framework. The framework itself was implemented by Martin Wagner as part of this Master thesis [Wag19]. The TOSCA extension shown here is taken from this implementation.

```
            agent:
                type: string
                description: address of agent to run the service on
                required: false
            first_in_chain:
                type: boolean
                default: false
                required: true
                description: is this microservice the first in the
                    chain (gets the input from the client)
            last_in_chain:
                type: boolean
                default: false
                required: true
                description: is this microservice the last in the
                    chain? (returns the output for the client)
            force_rebuild:
                type: boolean
                default: false
                required: false
                description: should this microservice container be
                    built anew?
            polling:
                type: boolean
                default: false
                required: false
                description: does this microservice support polling
                     for activity?
            new_instance:
                type: boolean
                default: false
                required: false
                description: should the creation of a new instance
                    for this microservice be enforced?
            alive_time:
                type: integer
                required: false
                description: overwrite the alive time of the
                    microservice itself
        capabilities:
            output:
                type: tosca.capabilities.microservice_output
                valid_source_types: [tosca.nodes.
                    chained_microservice]
        requirements:
            - input:
                capability: tosca.capabilities.microservice_output
                node: tosca.nodes.chained_microservice
                relationship: tosca.relationships.microservices.
                    output_input
                occurrences: [ 0, UNBOUNDED ]
```

```
###################################################################
# Relationship Type.
# A Relationship Type is a reusable entity that defines the type
# of one or more relationships between Node Types or Node
# Templates.
###################################################################
relationship_types:
    tosca.relationships.microservices.output_input:
        derived_from: tosca.relationships.Root
        valid_target_types: [ tosca.capabilities.
            microservice_output ]

###################################################################
# Capability Type.
# A Capability Type is a reusable entity that describes a kind of
# capability that a Node Type can declare to expose.
###################################################################
capability_types:
    tosca.capabilities.microservice_output:
        derived_from: tosca.capabilities.Root

###################################################################
# Data Type.
# A Datatype is a complex data type declaration which contains
# other complex or simple data types.
###################################################################
data_types:
    tosca.datatypes.microservice_io:
      derived_from: tosca.datatypes.Root
      type: string
      description: a datatype for defining the type of an input or
          output
      constraints:
        - valid_values: [ number, integer , float , string , text_file
           , image, list ]

    tosca.datatypes.microservice_inputs:
      derived_from: tosca.datatypes.Root
      type: list
      description: a list of constrained strings to define the
          inputs of a microservice
      entry_schema:
        type: tosca.datatypes.microservice_io

    tosca.datatypes.microservice_outputs:
      derived_from: tosca.datatypes.Root
      type: list
      description: a list of constrained strings to define the
          outputs of a microservice
      entry_schema:
        type: tosca.datatypes.microservice_io

    tosca.datatypes.semantic_description:
        derived_from: tosca.datatypes.Root
        description: semantic description for a microservice from
            the store
        properties:
            category:
                type: string
                required: true
            inputs:
                type: tosca.datatypes.microservice_inputs
                required: false
            outputs:
                type: tosca.datatypes.microservice_outputs
                required: false

##################################################################
# Group Type.
# Group Type represents logical grouping of TOSCA nodes that have
# an implied membership relationship and may need to be
# orchestrated or managed together to achieve some result.
##################################################################
group_types:
  tosca.groups.docker_bridge_network:
    derived_from: tosca.groups.Root
    description: Group type to set up a docker user—defined bridge
        network
    properties:
      name:
        type: string
        description: Name of this bridge network
```

APPENDIX D

TOSCA Description of the Word Count Service Chain[1]

```
tosca_definitions_version:
    tosca_simple_profile_for_microservice_chains_1_0_0

description: Description of the microservice chain used for the
    evaluation.

topology_template:
  node_templates:
    wordsplit:
      type: tosca.nodes.chained_microservice
      properties:
        store_id: 5d1f4c1c53b89fd219f084c8
        first_in_chain: false
        last_in_chain: false
        force_rebuild: false
        polling: false
      requirements:
        - input: echo1

    wordcount:
      type: tosca.nodes.chained_microservice
      properties:
        store_id: 5d1f4c2853b89fd219f084cc
        first_in_chain: false
        last_in_chain: false
        force_rebuild: false
        polling: false
      requirements:
        - input: wordsplit
```

[1] **Contribution statement:** I led the idea generation and design of the flexEdge framework. The framework itself was implemented by Martin Wagner as part of this Master thesis [Wag19]. The TOSCA description shown here is taken from this implementation.

```
echo1:
  type: tosca.nodes.chained_microservice
  properties:
    store_id: 5d1f4eac53b89fd219f084d6
    first_in_chain: true
    last_in_chain: false

echo2:
  type: tosca.nodes.chained_microservice
  properties:
    store_id: 5d1f4eac53b89fd219f084d6
    first_in_chain: false
    last_in_chain: true
  requirements:
    - input: wordcount
```

APPENDIX E

Detailed Execution Times of Microservices

This appendix provides the detailed execution times of the microservices, as evaluated in Section 7.6.2.a. Table E.1 lists the average (AVG), standard deviation (SD), minimum (MIN) and maximum (MAX) values for all evaluation conditions. All values are given in milliseconds.

TABLE E.1: EXECUTION TIMES OF MICROSERVICES

		Microservice		
		Object detection	Face detection	Word count
WiFi **MS-Store** **cold start**	AVG	7578.1	1199.6	888.53
	SD	164.61	52.34	47.72
	MIN	7300	1116	841
	MAX	8104	1350	1021
WiFi **MS-Store** **warm start**	AVG	1371	387.47	134.43
	SD	175.93	38.19	41.11
	MIN	1208	310	34
	MAX	2060	470	178
WiFi **Offload** **cold start**	AVG	10657.2	1194.77	899.37
	SD	255.05	48.38	67.59
	MIN	10143	1107	837
	MAX	11161	1295	1164
WiFi **Offload** **warm start**	AVG	6212.6	600.17	270.7
	SD	315.67	70.76	110.83
	MIN	5745	477	161
	MAX	7146	719	460

		Microservice		
		Object detection	Face detection	Word count
Celluar MS-Store cold start	AVG	8438.5	2472.2	1049.3
	SD	641.21	370.78	55.8
	MIN	7565	1944	945
	MAX	9957	3696	1211
Celluar MS-Store warm start	AVG	2470.5	1429.13	208.13
	SD	447.92	197.46	27.11
	MIN	1942	1203	156
	MAX	4016	2158	286
Celluar Offload cold start	AVG	46475.23	2747.43	1038.5
	SD	15628.61	777.2	67.7
	MIN	27095	1690	874
	MAX	84904	5077	1185
Celluar Offload warm start	AVG	32192.97	1584.97	324.83
	SD	10004.54	233.81	42.26
	MIN	21776	1285	249
	MAX	64972	2163	460

APPENDIX F

Pyomo ILP Model for Operator Placement

LISTING F.1: MODEL.PY SOURCE CODE FILE

```
from __future__ import division
from pyomo.environ import *
from itertools import product

def create_abstract_model():
    model = AbstractModel()

    model.OPERATORS = Set()
    model.NODES = Set()
    model.OPEDGES = Set(within=model.OPERATORS * model.OPERATORS)
    model.NODEEDGES = Set(within=model.NODES * model.NODES)

    model.operator_workload = Param(model.OPERATORS,within=
        ↪ NonNegativeIntegers)
    model.node_capacity = Param(model.NODES, within=
        ↪ NonNegativeIntegers)
    model.operator_datarate = Param(model.OPEDGES,within=
        ↪ NonNegativeIntegers)

    model.placement_cost = Param(model.OPERATORS, model.NODES,
        ↪ within=NonNegativeReals)
    model.d = Param(model.NODES * model.NODES,within=
        ↪ NonNegativeIntegers)
    model.b = Param(model.NODEEDGES,within=NonNegativeIntegers)

    model.pinnings = Set(within=model.OPERATORS * model.NODES)
    model.pinning_options = Set(within=model.OPERATORS * model.
        ↪ NODES)
    model.exclusions = Set(within=model.OPERATORS * model.NODES)
    model.colocations = Set(within=model.OPERATORS * model.
        ↪ OPERATORS)

    model.paths = Set(within=model.NODES * model.NODES * model.
        ↪ NODEEDGES)

    model.x = Var(model.OPERATORS, model.NODES, domain=Binary,
        ↪ within=Binary, initialize=0)
    model.y = Var(model.OPERATORS, model.OPERATORS, model.NODES,
        ↪ model.NODES, domain=Binary, within=Binary, initialize
        ↪ =0)

    model.wr = Param(within=NonNegativeReals, default=0.5)
    model.wd = Param(within=NonNegativeReals, default=0.5)

    def obj_expression(model):
        r = 0
        d = 0
        for o in model.OPERATORS:
            for n in model.NODES:
                r += (model.placement_cost[o,n] * model.x[o,n])
        for (i,j) in model.OPEDGES:
            for (u,v) in product(model.NODES,model.NODES):
                d += model.d[u,v] * model.y[i,j,u,v]
        return (model.wr * r) + (model.wd * d)
    model.OBJ = Objective(rule=obj_expression)

    def pinning_rule(model,o,u):
        return model.x[o,u] == 1
    model.PinningRule = Constraint(model.pinnings,rule=
        ↪ pinning_rule)

    def pinning_options(model,o):
        if not model.pinning_options:
            return Constraint.Feasible
        return sum(model.x[o,n] for (o,n) in model.pinning_options
            ↪ ) == 1
    model.PinningOptions = Constraint(model.OPERATORS, rule=
        ↪ pinning_options)

    def colocation_constraint(model,o,p,n):
        return model.x[o,n] == model.x[p,n]
    model.ColocationRule = Constraint(model.colocations, model.
        ↪ NODES, rule=colocation_constraint)

    def exclusion_rule(model,o,u):
        return model.x[o,u] == 0
    model.ExclusionRule = Constraint(model.exclusions,rule=
        ↪ exclusion_rule)

    def rule_capacity(model, u):
        return sum(model.operator_workload[o] * model.x[o,u] for o
            ↪ in model.OPERATORS) <= model.node_capacity[u]
    model.CapacityConstraint = Constraint(model.NODES, rule=
        ↪ rule_capacity)

    def rule_uniqueplacement(model, i):
        return sum(model.x[i,n] for n in model.NODES) == 1
    model.PlacementConstraint1 = Constraint(model.OPERATORS, rule=rule_uniqueplacement)

    def network_rule1(model,i,j,u):
        return sum(model.y[i,j,u,v] for v in model.NODES) == model.x[i,u]
    model.NetworkConstraint1 = Constraint(model.OPEDGES,model.NODES, rule=network_rule1)

    def network_rule2(model,i,j,v):
        return sum(model.y[i,j,u,v] for u in model.NODES) == model.x[j,v]
    model.NetworkConstraint2 = Constraint(model.OPEDGES,model.NODES, rule=network_rule2)

    def link_rule(model,a,b):
        subset_paths = []
        for u,v,c,d in model.paths:
            if (c,d) == (a,b):
                subset_paths.append(tuple((u,v)))
        if not subset_paths:
            return Constraint.Feasible
        else:
            return  sum(sum(model.operator_datarate[j,k] * model.y[j,k,u,v] for j,k in model.OPEDGES) for u,v in subset_paths ) <= model.b[a,b]
    model.LinkRule = Constraint(model.NODEEDGES, rule=link_rule)

    return model
```

APPENDIX G

Problem Sizes for the Operator Placement Evaluation

This appendix details how the input sizes for the evaluation of the operator placement heuristics (see Section 8.6) are constructed. Table G.1 shows how many operator graphs of each type (following the numbering a–m as in Figure 8.5) are contained in the input sizes g1–g11.

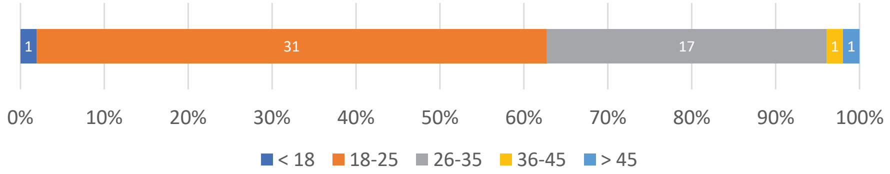
TABLE G.1: NUMBER OF OPERATOR GRAPHS PER INPUT SIZE

		Input sizes										
		gs1	gs2	gs3	gs4	gs5	gs6	gs7	gs8	gs9	gs10	gs11
Operator graphs	a	2	2	2	2	2	2	2	2	2	2	2
	b									1	1	1
	c							1	1	1	1	1
	d		1	1	1	1	1	1	1	1	1	1
	e	1	1	1	1	1	1	1	1	1	1	1
	f											1
	g				1	1	1	1	1	1	1	1
	h						1	1	1	1	1	1
	i								1	1	1	1
	j	1	1	1	1	2	2	2	2	2	2	2
	k			1	1	1	1	1	1	1	1	1
	l	1	1	1	1	1	1	1	1	1	1	1
	m										1	1

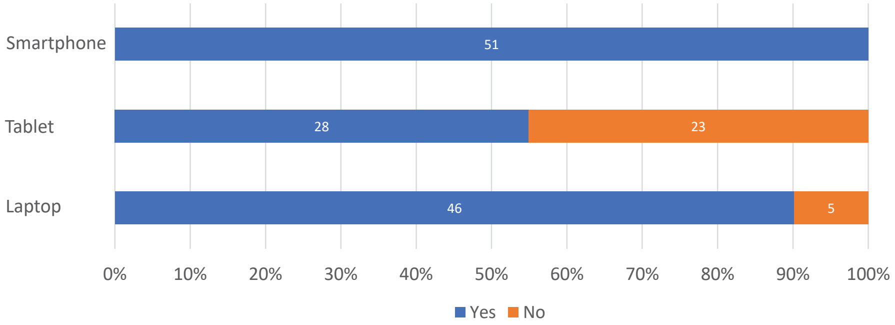
APPENDIX H

Questions of the Survey on Mobile Storage[1]

Q1: How old are you?

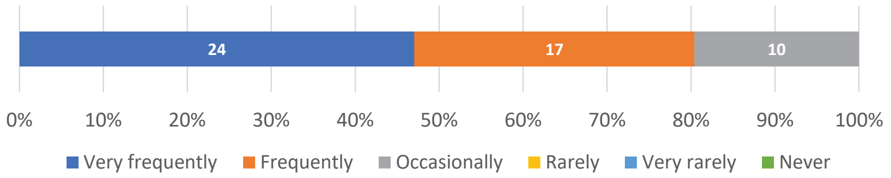


Q2: Which of these devices do you own?

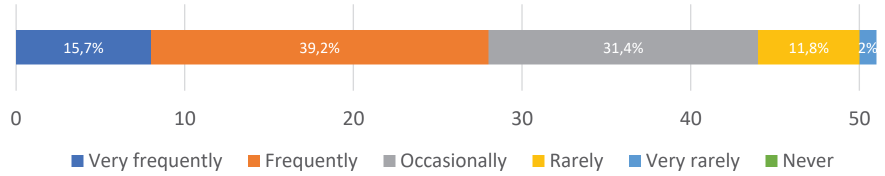


[1] **Contribution statement:** I led the design of the questionnaire and the analysis of the results. The survey itself was carried out by Nicolás Himmelmann as part of this Bachelor's thesis [Him17].

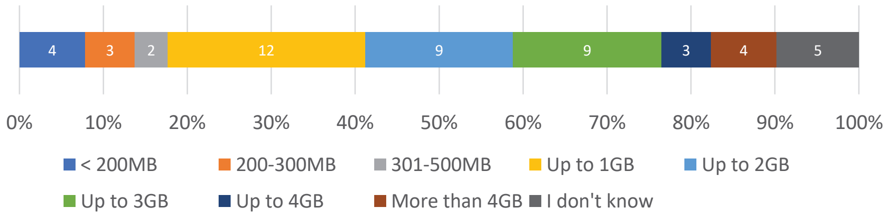
Q3: How often do you use your smartphone on an average day?



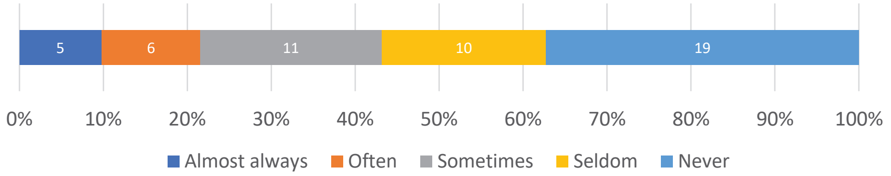
Q4: How often do you use your other mobile devices (e.g., laptop or tablet) on an average day?



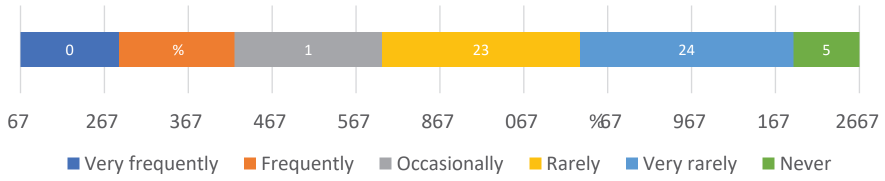
Q5: What is your monthly data plan?



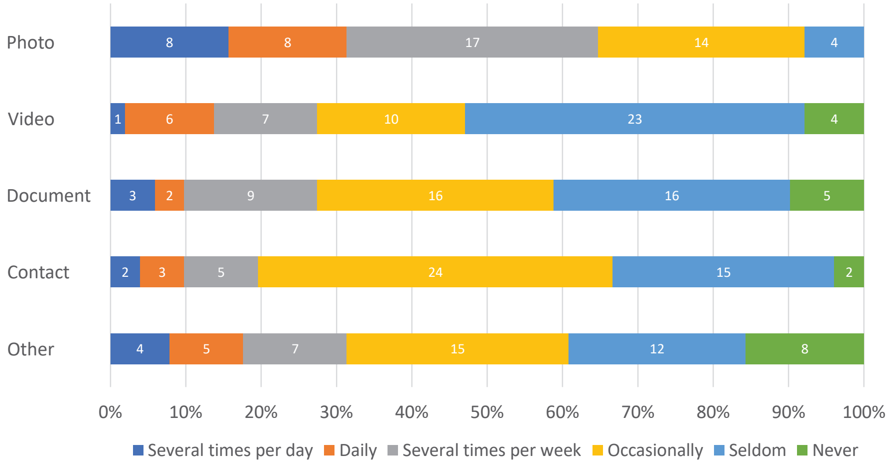
Q6: How often do you exceed this data plan?



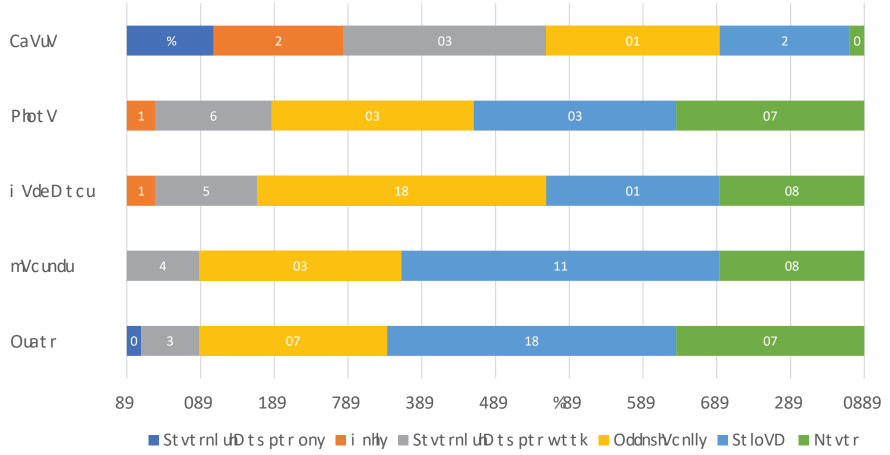
Q7: How often do you use public WiFi networks?



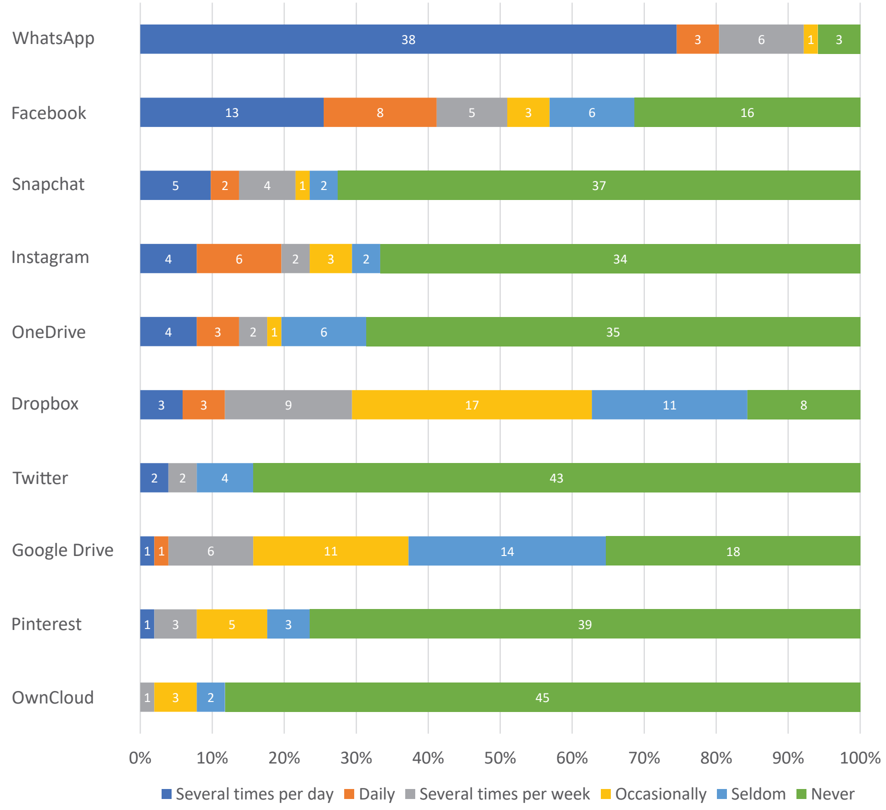
Q8: How often do you capture data of the following types with your mobile device?



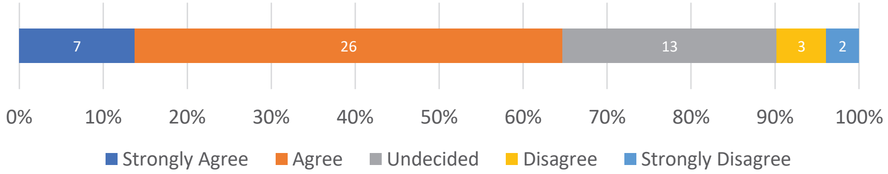
Q9: How often do you share data of the following types with your mobile device?



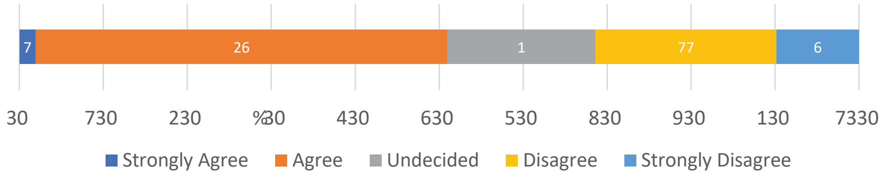
Q10: Which of the following services do you use?



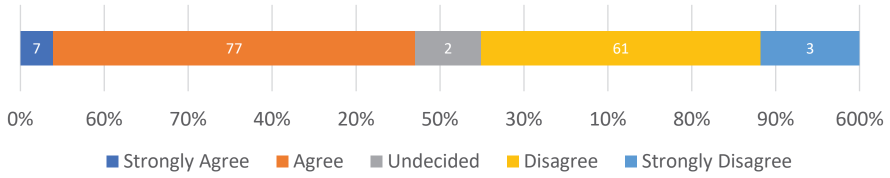
Q11: Which of these services I use depends on whether I want to share the data or store it for private use.



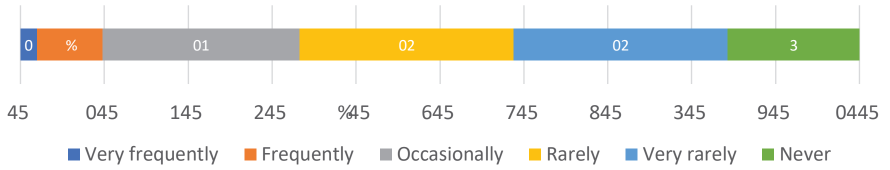
Q12: Which of these services I use depends on my current location.



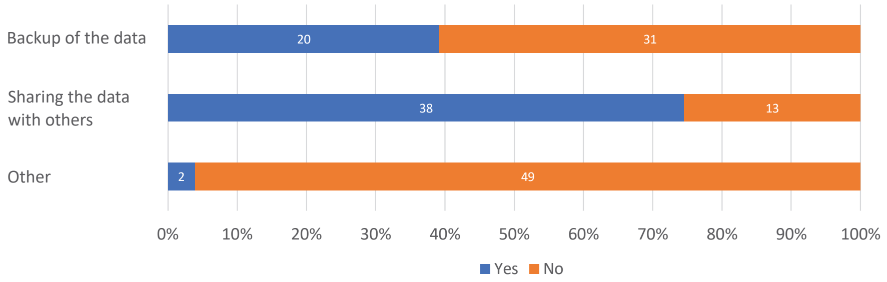
Q13: Which of these services I use depends on the time of the day or the day of the week.



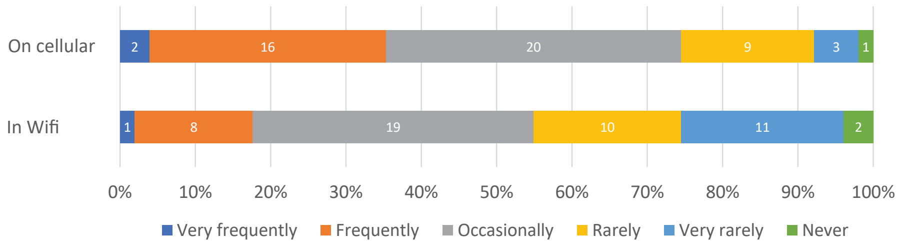
Q14: How often do you upload the same data (e.g., a photo or a document) to more than one service?



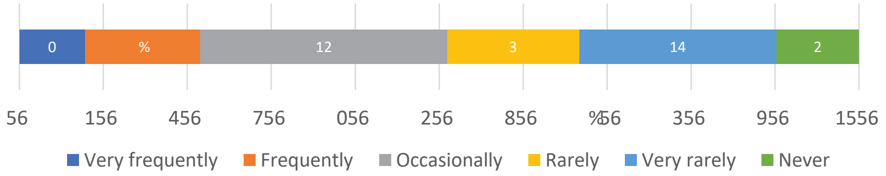
Q15: If so, for which purpose?



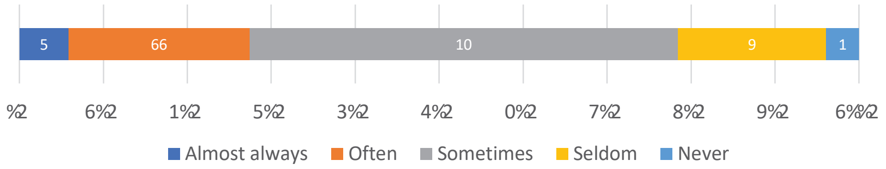
Q16: How often do you experience considerable delays when retrieving files in mobile networks?



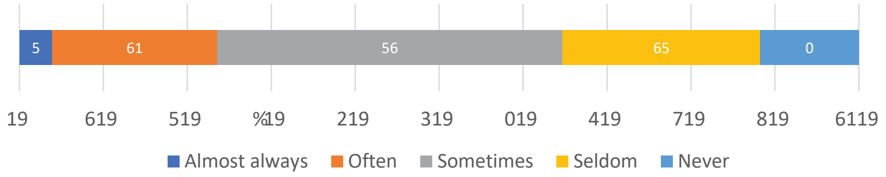
Q17: How often do you share data at an event?



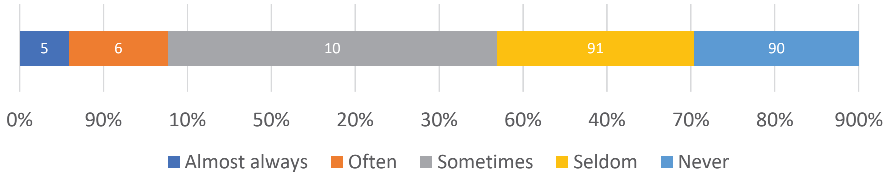
Q18: Did you experience times where the network was overloaded at an event?



Q19: Do you retrieve data related to this event (e.g., photos/videos) while you are there?



Q20: How often do you share data with others that are also present at the same event?



APPENDIX I

Implementation Details of vStore[1]

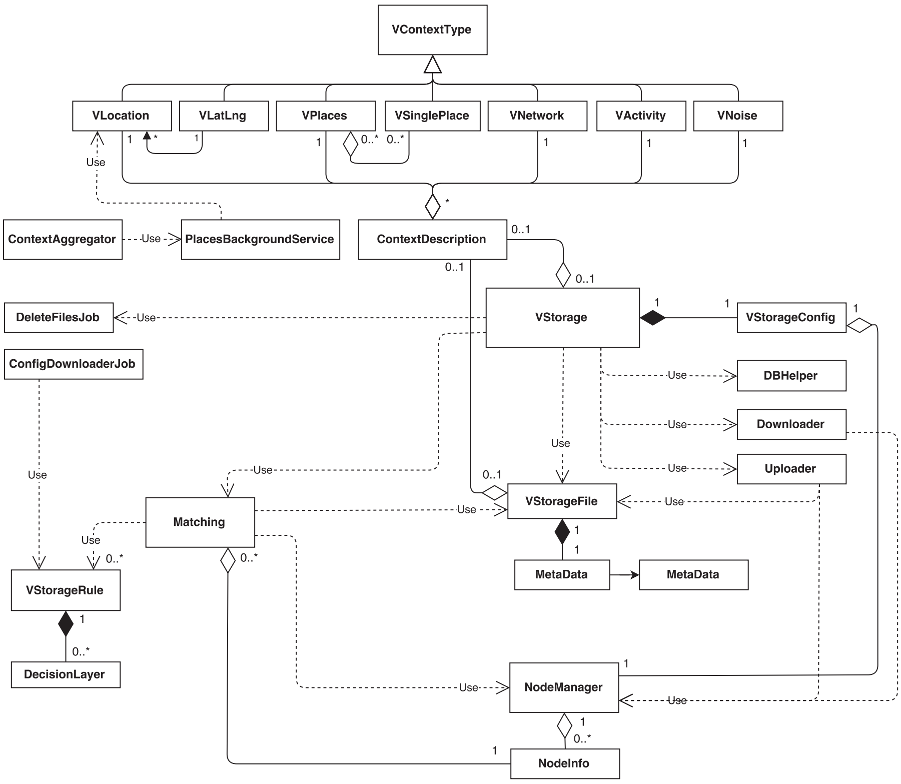


FIGURE I.1: CLASS DIAGRAM OF THE VSTORE FRAMEWORK

[1] **Contribution statement:** I led the idea generation and design of the vStore framework. The framework itself was implemented by Nicolás Himmelmann as part of this Bachelor's thesis [Him17]. The figures in this appendix are taken from this thesis.

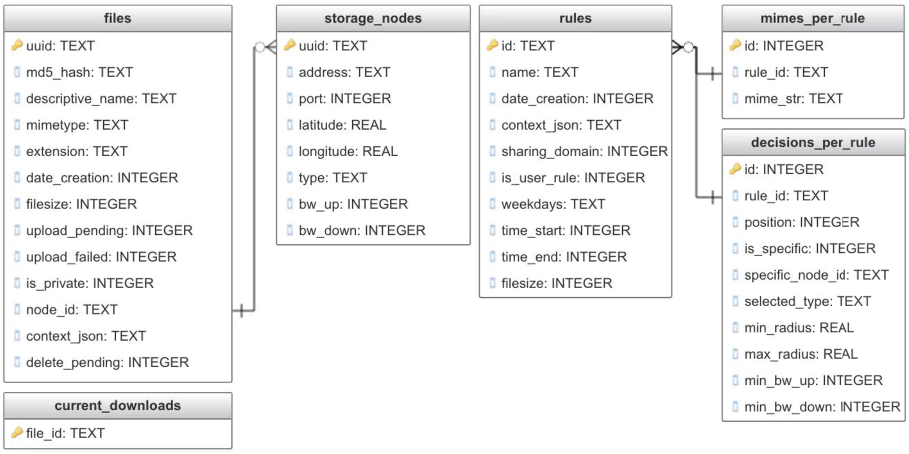


FIGURE I.2: DATABASE SCHEME OF THE SQLITE DATABASE ON THE MOBILE CLIENT

APPENDIX J

Example Class Diagram of an Adaptable Microservice[1]

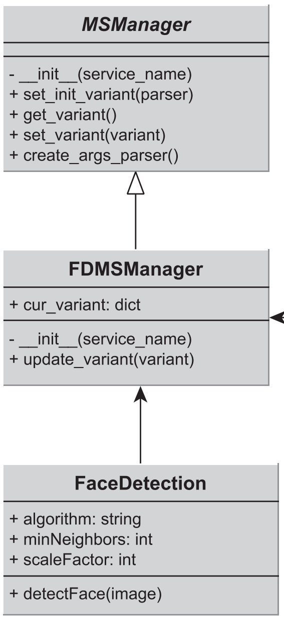


FIGURE J.1: UML CLASS DIAGRAM OF THE ADAPTABLE FACE DETECTION MICROSERVICE

[1] **Contribution statement:** I led the idea generation and design of the system for adaptable microservices. The student assistant Karolis Skaisgiris was involved in developing a prototype implementation. The class diagram shown in this appendix is taken from this implementation.

APPENDIX K

Wissenschaftlicher Werdegang des Verfassers

2007–2013	Studium der Informatik an der TU Darmstadt Abschluss: *Bachelor of Science*
2013–2015	Studium der Informatik an der TU Darmstadt Abschluss: *Master of Science*
2016–2020	Wissenschaftlicher Mitarbeiter am Fachgebiet Telekooperation des Fachbereichs Informatik an der TU Darmstadt