Philipps Universität Marburg

Situation-aware Edge Computing

Dissertation

zur Erlangung des Doktorgrades der Naturwissenschaften
(Dr. rer. nat.)

dem Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
vorgelegt von

M.Sc.
Artur Sterz
geboren in Nowodolinka

Marburg, im Juni 2022

Vom Fachbereich Mathematik und Informatik der Philipps-Universität Marburg
(Hochschulkennziffer 1180) als Dissertation am 27.06.2022 angenommen.

1. Gutachter: Prof. Dr.-Ing. Bernd Freisleben, Philipps-Universität Marburg
2. Gutachter: Prof. Dr. Oliver Hinz, Goethe-Universität Frankfurt am Main

Tag der Einreichung am 20.06.2022.
Tag der mündlichen Prüfung am 15.09.2022.

Eidesstattliche Erklärung

Ich versichere, dass ich meine Dissertation selbstständig, ohne unerlaubte Hilfe angefertigt und mich dabei keiner anderen als der von mir ausdrücklich bezeichneten Quellen und Hilfen bedient habe. Die Dissertation wurde in der jetzigen oder einer ähnlichen Form noch bei keiner anderen Hochschule eingereicht und hat noch keinen sonstigen Prüfungszwecken gedient. Des Weiteren wurde noch keine Promotion an einer anderen Universität oder der Philipps-Universität Marburg versucht.

Datum

Unterschrift

Abstract

Future wireless networks must cope with an increasing amount of data that needs to be transmitted to or from mobile devices. Furthermore, novel applications, e.g., augmented reality games or autonomous driving, require low latency and high bandwidth at the same time. To address these challenges, the paradigm of edge computing has been proposed. It brings computing closer to the users and takes advantage of the capabilities of telecommunication infrastructures, e.g., cellular base stations or wireless access points, but also of end user devices such as smartphones, wearables, and embedded systems. However, edge computing introduces its own challenges, e.g., economic and business-related questions or device mobility. Being aware of the current situation, i.e., the domain-specific interpretation of environmental information, makes it possible to develop approaches targeting these challenges.

In this thesis, the novel concept of situation-aware edge computing is presented. It is divided into three areas: situation-aware infrastructure edge computing, situation-aware device edge computing, and situation-aware embedded edge computing. Therefore, the concepts of situation and situation-awareness are introduced. Furthermore, challenges are identified for each area, and corresponding solutions are presented. In the area of situation-aware infrastructure edge computing, economic and business-related challenges are addressed, since companies offering services and infrastructure edge computing facilities have to find agreements regarding the prices for allowing others to use them. In the area of situation-aware device edge computing, the main challenge is to find suitable nodes that can execute a service and to predict a node's connection in the near future. Finally, to enable situation-aware embedded edge computing, two novel programming and data analysis approaches are presented that allow programmers to develop situation-aware applications.

To show the feasibility, applicability, and importance of situation-aware edge computing, two case studies are presented. The first case study shows how situation-aware edge computing can provide services for emergency response applications, while the second case study presents an approach where network transitions can be implemented in a situation-aware manner.

Deutsche Zusammenfassung

Zukünftige drahtlose Netze müssen immer größere Datenmengen bewältigen, die an oder von mobilen Geräten übertragen werden müssen. Außerdem erfordern neue Anwendungen, z.B. Augmented-Reality Spiele oder autonomes Fahren, niedrige Latenzzeiten und gleichzeitig hohe Bandbreiten. Um diese Herausforderungen zu bewältigen, wurde das Paradigma des Edge Computing vorgeschlagen. Es bringt die Verarbeitungskapazitäten näher an die Nutzer heran und nutzt die Möglichkeiten von Telekommunikationsinfrastrukturen, z.B. Mobilfunk-Basisstationen oder drahtlose Zugangspunkte, aber auch von Endnutzergeräten wie Smartphones, tragbaren Geräten und eingebetteten Systemen. Das Edge Computing hat jedoch eigene Herausforderungen, z.B. wirtschaftliche und geschäftsbezogene Fragen oder Mobilität. Die Kenntnis der aktuellen Situation, d.h. die domänenspezifische Interpretation von Umgebungsinformationen, ermöglicht die Entwicklung von Ansätzen, die diese Herausforderungen adressieren.

In dieser Arbeit wird das neue Konzept des situationsbewussten Edge Computing vorgestellt. Es ist in drei Bereiche unterteilt: situationsbewusstes Infrastruktur Edge Computing, situationsbewusstes Geräte Edge Computing Computing und situationsbewusstes eingebettetes Edge Computing. Daher werden die Konzepte Konzepte der Situation und des Situationsbewusstseins eingeführt. Außerdem werden für jeden Bereich Herausforderungen identifiziert und entsprechende Lösungen vorgestellt. Im Bereich des situationsbewussten Infrastruktur Edge Computing werden wirtschaftliche und geschäftliche Herausforderungen angesprochen, da Unternehmen, die Dienste und und Infrastruktur Edge Computing anbieten, Vereinbarungen über die Preise für die Nutzung durch andere treffen müssen. Im Bereich des situationsbewussten Geräte Edge Computing besteht die größte Herausforderung darin, geeignete Knoten zu finden, die einen Dienst ausführen können, und die Verbindungen eines Knotens in der nahen Zukunft vorherzusagen. Um schließlich situationsbewusstes eingebettetes Edge Computing zu ermöglichen, werden zwei neue Programmier- und Datenanalyseansätze vorgestellt, die es Programmierern ermöglichen, situationsabhängige Anwendungen zu entwickeln.

Um die Durchführbarkeit, Anwendbarkeit und Bedeutung des situationsbewussten Edge Computings zu zeigen, werden zwei Fallstudien präsentiert. Die erste Fallstudie zeigt, wie situationsbezogenes Edge Computing Dienste für Krisenanwendungen bereitstellen kann, während die zweite Fallstudie einen Ansatz vorstellt, bei dem Netzwerk-Transitionen situationsbewusst implementiert werden können.

Acknowledgements

First and foremost, I want to express my appreciation and gratitude to my mentor and doctoral advisor Prof. Dr. Bernd Freisleben. He supported me not only during the preparation of my dissertation and my time as a doctoral student, but also during my years as a student.

I would also like to thank Prof. Dr. Oliver Hinz from the Goethe-Universität Frankfurt am Main: first, for taking the time to review this thesis and, second, for giving me the opportunity to work together with him and his research associates in the DFG SFB 1053.

I would like to thank my proofreaders Jonas Höchst and Patrick Lampe.

I also want to thank all my colleagues, collaborators, and advisors in the Distributed Systems Group at Marburg who assisted me as co-authors on scientific papers, teaching, or other project-related tasks (in alphabetical order): Nik Korfhage, Matthias Leinweber, Dr. Markus Mühling, Alvar Penning, Nils Schmidt, Stefan Schulz, Michael Schwarz, Dr. Roland Schwarzkopf, Markus Sommer, Christoph Thelen, and Christian Uhl.

Furthermore, I would like to express my gratitude to the DFG SFB 1053 MAKI, which provided financial support for my PhD and this thesis as well as an excellent environment for research, exchange of interests, and collaboration. Here, I would like to thank especially my co-authors of scientific publications, who supported me with expertise in their respective fields and without whom many works would not have been possible: Prof. Dr. Ralf Steinmetz, Prof. Dr. Mira Mezini, Prof. Dr. Oliver Hinz, Prof. Dr. Anja Klein, Prof. Dr. Matthias Hollick, Prof. Dr. Bernhard Seeger, Prof. Dr. Guido Salvaneschi, Dr. Ragnar Mogk, Matthias Eichholz, Dr. Sabrina Kloß, Dr. Katharina Keller, Dr. Alexander Frömmgen, Dr. Denny Stohr, Dr. Michael Körber, Dr. Pascal Weisenburger and Franz Kuntke. Special thanks to Dr. Patrick Felka for providing insightful information in the field of business informatics and Bernd Simon for discussions and explanations about game theory. Both were excellent collaborators with whom working on scientific publications was a lot of fun and very enlightening.

Of particular importance during my time as a PhD student were my colleagues Dr. Lars Baumgärtner, Dr. Pablo Graubner, Jonas Höchst, and Patrick Lampe. Dr. Lars Baumgärtner and Dr. Pablo Graubner supported me significantly in my beginnings and were also not afraid to criticize my work to push me to do my best. Jonas Höchst and Patrick Lampe, on the other hand, were not only colleagues, but over time became friends who not only provided professional advice, but also offered emotional support. Many thanks to both of you. Without you all this would not have been possible.

Last but definitely not least, I have to thank my mother and my wife. Without my mother's encouragement and her tireless efforts to motivate me to do my best, I would not have decided to pursue academia. My wife supported me all the way and stood by me through all the decisions and hard times. I could not have done it without her encouragement, support and love. Thank you.

My Contributions

Research in computer science is ultimately a collaborative effort that requires the interaction of many contributors. Scientific ideas are generated collaboratively, novel concepts are conceived and formalized by bringing in different expertise, implementations are often jointly developed, and results are collaboratively discussed and interpreted. In addition, students play an important role in implementing concepts or evaluating them. As indicated in the Acknowledgements on page xi, this is also evident in this work. Therefore, it is impossible to precisely pinpoint the input of individual contributors. Since this thesis is based on original publications and parts are given verbatim in places, I try to emphasize my contributions of the individual chapters in this section.

Chapter 2 provides explanations and backround information for concepts, technologies, and terms. Parts of Subsection 2.1.2 are based on publication [191]. Chapter 3 introduces the novel concept of situation-aware edge computing. The contributions made in this chapter are solely my own ideas.

Chapter 4 is based on two publications, namely publication [236] in Section 4.2 and publication [81] in Section 4.3. The concept, system model, and evaluation presented in Section 4.2 was designed collaboratively by Dr. Patrick Felka, Dr. Sabrina Klos, Bernd Simon and me. The presented Nash bargaining solution was formalized by Dr. Sabrina Klos, whereas the novel solution of iterative bargaining was conceptualized and formalized by Bernd Simon and me. The evaluation was provided by Dr. Patrick Felka. Dr. Prof. Anja Klein, Prof. Dr. Oliver Hinz, and Prof. Dr. Bernd Freisleben reviewed the paper and suggested improvements. The Concept and the design in Section 4.3 was provided by Dr. Patrick Felka, Dr. Katharina Keller and me. Data preparation was implemented by Dr. Patrick Felka and I, and Dr. Patrick Felka contributed the evaluation. Prof. Dr. Oliver Hinz and Prof. Dr. Bernd Freisleben provided feedback and improvements with respect to the writing.

Publication [233] in Section 5.2 and publication [112] are the basis of Section 5.3 in Chapter 5. The concept presented in Section 5.2 is solely my contribution, with Dr. Lars Baumgärtner providing feedback. The proof-of-concept was implemented by myself, while Jonas Höchst suggested improvements. The evaluation was done by Jonas Höchst and me. Patrick Lampe and Prof. Dr. Bernd Freisleben reviewed the paper and gave feedback regarding the writing. Jonas Höchst, Dr. Alexander Frömmgen, Dr. Denny Stohr, and myself jointly developed the concept presented in Section 5.3. The design, implementation, and evaluation of the data collection, data preparation, and implementation of the machine learning model was done by Jonas Höchst, while I aided the design. My main contribution was the design, implementation, and execution of the evaluation, while Jonas Höchst supported me with helpful feedback. Prof. Dr. Ralf Steinmetz and Prof. Dr. Bernd Freisleben suggested improvements of the paper.

Chapter 6 presents work that is based on publication [235] in Section 6.2 and publication [104] in Section 6.3. The concept in Section 6.2 was developed collaboratively by Dr. Pablo Graubner and me. The design and implementation of the programming language frontend and the corresponding compiler was done by Dr. Ragnar Mogk, Matthias Eichholz, and Dr. Pascal Weisenburger, while I supplied domain knowledge to direct the design of the language. Prof. Dr. Matthias Hollick and me collaborated on developing the use cases and

resolving implementation issues. Prof. Dr. Matthias Hollick's research group developed position independent code modules for the firmware patching framework Nexmon. I implemented the use cases and conducted the experiments. Dr. Lars Baumgärtner, Prof. Dr. Mira Mezini, Prof. Dr. Matthias Hollick, and Prof. Dr. Bernd Freisleben reviewed, commented, and edited the paper before submission. The concept presented in Section 6.3 is genuinely the work of Dr. Pablo Graubner. Christoph Thelen implemented parts of the CQL frontend and prototyped the eBPF mode, Dr. Michael Körber contributed the mode selection algorithm. I implemented the use case for gathering and analyzing mobility data and conducted most of the experiments. Prof. Dr. Bernhard Seeger, Prof. Dr. Guido Salvaneschi, Prof. Dr. Mira Mezini, and Prof. Dr. Bernd Freisleben reviewed, commented, and edited the publication before submission.

Chapter 7 in Section 7.2 combines parts that were published in the publications [23, 104, 233, 235]. The presented approach in Section 7.2.1 was designed by Dr. Lars Baumgärtner and me, whereas the implementation and evaluation was genuinely my work. The design and implementation of Section 7.2.2 was solely my work, while the evaluation was done collaboratively with Jonas Höchst. Section 7.2.3 was designed, implemented, and evaluated entirely by myself. Section 7.3 of Chapter 7 presents an approach that was published in publication [234]. The design, implementation, and evaluation is genuinely my work, with Dr. Lars Baumgärtner providing feedback on all points. Dr. Ragnar Mogk, Prof. Dr. Mira Mezini, and Prof. Dr. Bernd Freisleben reviewed, commented, and edited the manuscript.

Contents

1 Introduction

Future wireless networks will face many challenges due to applications that are not suitable for current wireless networks. First, according to Ericsson, mobile data traffic, i.e., data sent through the Internet originating from or destined to mobile devices, already increased by about 650%, from 10 exabyte to 75 exabyte in the years 2016 to 2021 and will increase even further about five times, from 75 exabyte to 375 exabyte by 2027 [241]. All of this data not only has to be transferred, but also needs to be processed. Second, new applications with special needs and requirements are currently being developed. Augmented Reality (AR) applications, multiplayer online games, autonomous driving, remote health applications, smart homes, and industry automation require low latency and high bandwidth at the same time.

However, end user devices like notebooks, tablets, mobile phones, wearables, and embedded devices in the area of the Internet of Things (IoT) offer interesting computing capabilities for (pre-)processing data near the users. Furthermore, routers, access points, and base stations of cellular networks can execute network functionality. New computing paradigms were introduced in the recent past that aim to bring computation and storage closer to users by relocating computations and data to the edge of the network, i.e., edge computing, fog computing or mobile cloud computing bring great benefits in terms of latency and throughput. In particular, edge computing as an alternative to cloud computing has emerged as a promising approach by utilizing cellular base stations, wireless access points, end user devices, IoT devices, and embedded systems as self-contained computing platforms.

However, edge computing also has its own challenges. For example, providing computation facilities near cellular base stations introduces economic and business-related challenges, since these facilities have to be operated and maintained. Furthermore, the devices used for edge computing are connected wirelessly (e.g., Wi-Fi, 4G/5G, LoRa, etc.) either with each other in a peer-to-peer manner or to centrally organized base stations or access points, introducing the need to cope with mobile devices, e.g., by developing handover mechanisms or network paradigms resilient to intermittent connectivity. In addition, new programming paradigms are required, since traditional imperative programming languages are error-prone and ill-suited for implementing novel data-driven applications tailored for edge computing applications.

To meet the challenge of rapidly increasing data volumes and the associated processing functionality, future computing paradigms must perceive their environment and comprehend the perceived information to derive the current situation to be able to adapt accordingly. The resulting awareness of their current situation will make it possible to develop applications, e.g., in the area of emergency response that can assist and adapt themselves during emergencies. Transitions, i.e., adaptation between similar mechanisms at runtime to increase the quality of service, can be implemented with the help of situation-awareness. But also the challenges of

the individual areas of edge computing, such as identifying suitable targets for a service to be placed based on the current situation or dealing with devices' mobility can be addressed.

The concept of perceiving one's environment, comprehending the perceived information, and thereby gaining domain-specific knowledge is called situation-awareness. In this thesis, it is argued that situation-aware edge computing brings forth novel applications to cope with the above challenges. In contrast to the concept of context, where only environmental information is analyzed in order to make action decisions based on it, the situation concept goes one step further. It is not only about analyzing the environment, but also about interpreting it and arriving at an understanding of the environment that is specific to the system or entity and contains its own interpretation. To make the difference clear, a short example will be given. In an emergency, a variety of information is gathered, such as data from early warning systems, analyses and assessments from authorities, imagery from satellites and so on. This information represents the context. However, different rescue helpers interpret this context information differently, according to their domain. Firefighters, for example, interpret the data in terms of fires, while paramedics interpret the context in terms of people to be rescued. The context is the same for both firefighters and paramedics, namely an emergency. However, the interpretation and the information that is important to them leads to different situations.

1.1 Problem Statement

The goal of this thesis is to facilitate situation-aware edge computing in the areas of infrastructure edge, device edge, and embedded edge. It provides an economic model and service usage prediction for situation-aware infrastructure edge computing, a protocol for offloading tasks and avoiding connection losses in situation-aware device edge computing, and novel analysis and programming methods for situation-aware embedded edge computing.

The first research question is how providers of services and providers of infrastructure edge facilities can find an agreement with respect to the price for utilizing infrastructure edge facilities. The two challenges in this area are (i) the problem of incomplete information, i.e., both parties are not able or willing to share all their private business information with each other, and (ii) to predict how many users will use the service, which is required to estimate the cost for utilizing the infrastructure edge facilities.

The second research question is how to decide which device should be used to offload a process to in the area of device edge and how to predict whether a wireless connection will be lost due to the nodes' mobility. It is crucial to identify nodes that are able to execute a process to increase the number of successful offloadings. However, it is not possible to determine the capabilities of the node from outside, i.e., the offloading node has no chance to find nodes able to execute a process.

The third research question is how to enable programmers of embedded edge applications to effectively and easily implement situation-aware applications. Programming paradigms used for embedded edge are usually imperative and rely on the C language. But C, similar to other imperative languages, is not well suited to implement applications that mainly process streams of data due to their data handling mode, where developers have to keep track of data references, allocation, and scoping.

1.2 Contributions of this Thesis

In this thesis, the following contributions are presented.

The novel approach of situation-aware edge computing is proposed. It makes use of the concepts of situation and situation-awareness that allow decision-making based on the perceived environmental information, comprehended to gather domain-specific understanding of the environment. Furthermore, situation-awareness is expanded to include incomplete information, an aspect that takes missing, inaccurate, or uncertain information, which may result from untrustworthy parties, faulty sensors, or inaccurate measurements, into account.

To achieve the goal of situation-aware edge computing, it is divided into three areas, (i) situation-aware infrastructure edge computing, (ii) situation-aware device edge computing, and (iii) situation-aware embedded edge computing. Each of these areas contains its own set of challenges and open research questions.

In the first area, situation-aware infrastructure edge computing, the main open research question is how to enable stakeholders of the Internet, i.e., companies and institutions offering services, providing compute- and storage resources and operating networks, to make economic decisions using their situation, which is built on information of their users, of other stakeholders but also of themselves. This is presented in two parts:

- A novel iterative bargaining approach between two stakeholders for nearly optimal service placement in infrastructure edge scenarios with respect to social cost despite incomplete information is proposed. By being aware of their situation through using publicly available information like price lists or scientific papers about pricing, both stakeholders project their future situation with respect to their own cost and the cost of respective other stakeholders and use this projection to decide where to place cloudlets finding a nearly optimal solution.
- A case study based on the AR game Ingress highlights metrics and measures for situation-aware mobile AR applications. In the course of the analysis, the relationship between the service provider's situation and the number of service usages is investigated. The results show that more than 84% of the variation of the number of in-game actions can be explained by being aware of the situation. This shows the enormous influence of the situation on the number of in-game actions.

The second area, situation-aware device edge computing, explores in two parts how situation-aware offloading and connection loss prediction can be realized:

- OPPLOAD, a novel situation-aware framework for offloading computational workflows in opportunistic networks, where so-called workers announce their capabilities and available resources that are perceived and comprehended to gather their situation by clients, i.e., the offloading devices. Clients then can project their future situation to decide on which worker a task should be offloaded to. This approach speeds up workflow execution and spreads the load fairly on spatially close but powerful workers, which increases the rate of successful offloadings significantly. Due to the workers announcing their information, clients can deal with incomplete information.
- A novel situation-aware approach Wi-Fi connection loss prediction to perform seamless vertical Wi-Fi/cellular handovers based on perceived sensor information on mobile devices is presented that predicts connection losses 15 seconds ahead using a machine

learning approach. Using this approach mobile devices are able to comprehend their situation with respect to the connection status and successfully project whether the connection is lost. This leads to the ability to decide whether a second connection should be established to provide seamless connections in device edge environments.

The third area, situation-aware embedded edge computing, introduces two approaches that enable situation-aware applications:

- With ReactiFi, a domain-specific language, programmability of embedded devices is facilitated. Programmers use a high-level reactive programming language to perceive environmental information, comprehend the device's situation and make decisions based on the projection of future situations. Advantages are its scheduling and memory usage. By comparing two implementations of a case study implemented in C and ReactiFi, the programming benefits of ReactiFi compared to C are discussed.
- Multimodal CEP is a novel approach to process streams of events on embedded devices. In multimodal CEP, queries are formulated in a high-level language, which are then broken up, and the most adequate execution mode for the involved CEP operators is selected. A multimodal CEP engine for Android devices is presented. It includes three novel execution environments for CEP operators in the operating system (based on Berkeley Packet Filters), on the Wi-Fi chip (based on the Nexmon firmware patching framework), and for a custom sensor hub, leading to significant power savings.

Finally, the concept of situation-aware edge computing is applied to two case studies.

- The novel idea of situation-aware edge computing for emergency response applications is introduced. It is identified that by applying the concept of situation-awareness to emergency response applications based on infrastructure edge, device edge and embedded edge approaches. Rescue helpers and people affected by emergencies are provided with useful information and applications that help to handle emergencies.
- DTN-RPC, a new approach to provide RPCs in device edge environments, is presented. DTN-RPC uses the concept of transitions to significantly increase the rate of successful RPCs. Further, to decide which transport protocol to use for the RPCs, i.e., to implement transitions, both client and server comprehend their situation with respect to their network state based on perceived information like available mobile devices using network lookups so that it can be decided which transport protocol will be used for the call or the transmission of the result. Finally, servers use a situation-aware approach where information from their sensors but also predicates from clients are used to comprehend their situation based on their ability to execute the called procedure.

During the work on this thesis, the following papers were published:

1. Lars Baumgärtner, Paul Gardner-Stephen, Pablo Graubner, Jeremy Lakeman, Jonas Höchst, Patrick Lampe, Nils Schmidt, Stefan Schulz, **Artur Sterz** and Bernd Freisleben. "An Experimental Evaluation of Delay-Tolerant Networking with Serval." In: *Proceedings of the Global Humanitarian Technology Conference (GHTC)*, pp. 70 – 79, IEEE, 2016

2. **Artur Sterz**, Lars Baumgärtner, Ragnar Mogk, Mira Mezini and Bernd Freisleben. "DTN-RPC: Remote Procedure Calls for Disruption-Tolerant Networking." In: *Proceedings of the IFIP Networking Conference (IFIP Networking)*, pp. 1 – 9, IEEE, 2017

3. 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: *Proceedings of the 17th International Conference on Mobile and Ubiquitous Multimedia (MUM)*, pp. 25 – 35, ACM, 2018

4. Pablo Graubner, Christoph Thelen, Michael Körber, **Artur Sterz**, Guido Salvaneschi, Mira Mezini, Bernhard Seeger and Bernd Freisleben. "Multimodal Complex Event Processing on Mobile Devices." In: *Proceedings of the 12th International Conference on Distributed and Event-Based Systems (DEBS)*, pp. 112 – 123, ACM, 2018

5. Patrick Felka, **Artur Sterz**, Oliver Hinz and Bernd Freisleben. "Using Social Media to Estimate the Audience Sizes of Public Events for Crisis Management and Emergency Care." In: *Proceedings of the International Conference on Smart Health (ICSH)*, pp. 77 – 89, Springer, 2018

6. **Artur Sterz**, Lars Baumgärtner, Jonas Höchst, Patrick Lampe and Bernd Freisleben. "OPPLOAD: Offloading Computational Workflows in Opportunistic Networks." In: *Proceedings of the 44th Conference on Local Computer Networks (LCN)*, pp. 381 – 388, IEEE, 2019

7. Jonas Höchst, **Artur Sterz**, Alexander Frömmgen, Denny Stohr, Ralf Steinmetz and Bernd Freisleben. "Learning Wi-Fi Connection Loss Predictions for Seamless Vertical Handovers Using Multipath TCP." In: *Proceedings of the 44th Conference on Local Computer Networks (LCN)*, pp. 18 – 25, IEEE, 2019 (*Best Paper Award*)

8. Lars Baumgärtner, Jonas Höchst, Patrick Lampe, Ragnar Mogk, **Artur Sterz**, Pascal Weisenburger, Mira Mezini and Bernd Freisleben. "Smart Street Lights and Mobile Citizen Apps for Resilient Communication in a Digital City." In: *Proceedings of the Global Humanitarian Technology Conference (GHTC)*, pp. 1 – 8, IEEE, 2019

9. Alvar Penning, Lars Baumgärtner, Jonas Höchst, **Artur Sterz**, Mira Mezini and Bernd Freisleben. "DTN7: An Open-Source Disruption-Tolerant Networking Implementation of Bundle Protocol 7." In: *Proceedings of the International Conference on Ad-Hoc Networks and Wireless (AdHoc-Now)*, pp. 196 – 209, Springer, 2019

10. Jonas Höchst, Lars Baumgärtner, Franz Kuntke, Alvar Penning, **Artur Sterz** and Bernd Freisleben. "LoRa-based Device-to-Device Smartphone Communication for Crisis Scenarios." In: *Proceedings of the 17th International Conference on Information Systems for Crisis Response and Management (ISCRAM)*, pp. 996 – 1011, ISCRAM, 2020

11. **Artur Sterz**, Matthias Eichholz, Ragnar Mogk, Lars Baumgärtner, Pablo Graubner, Matthias Hollick, Mira Mezini and Bernd Freisleben. "ReactiFi: Reactive Programming of Wi-Fi Firmware on Mobile Devices." In: *The Art, Science, and Engineering of Programming*, Volume 5, Number 2, pp. 1 – 37, AOSA, 2020

12. **Artur Sterz**, Patrick Felka, Bernd Simon, Sabrina Klos, Anja Klein, Oliver Hinz and Bernd Freisleben. "Multi-Stakeholder Service Placement via Iterative Bargaining with Incomplete Information." In: *Transactions on Networking*, Early Access, pp. 1 – 16, IEEE/ACM, 2022

13. Jonas Höchst, Lars Baumgärtner, Franz Kuntke, Alvar Penning, **Artur Sterz**, Markus Sommer and Bernd Freisleben. "Mobile Device-to-Device Communication for Crisis Scenarios Using Low-cost LoRa Modems" In: *Disaster Management and Information Technology*, to appear, Springer, 2022

14. Markus Sommer, Jonas Höchst, **Artur Sterz**, Alvar Penning, and Bernd Freisleben. "ProgDTN: Programmable Disruption-tolerant Networking" In: *Proceedings of the 10th International Conference on Networked Systems (NETYS)*, to appear, Springer, 2022

Furthermore, the following software artifacts were co-developed and released under permissive open-source licences during the work on this thesis:

1. **DTN-RPC**: This is a C-based reference implementation of the remote procedure call protocol presented in *DTN-RPC: Remote Procedure Calls for Disruption-Tolerant Networking* [234]. It uses Serval [97] as the disruption-tolerant networking component. Available at `https://github.com/adur1990/DTN-RPC`.

2. **OPPLOAD**: This is an implementation of the computational workflow offloading protocol presented in *OPPLOAD: Offloading Computational Workflows in Opportunistic Networks* [233]. It is written in Python and uses Serval [97] to provide disruption-tolerant networking capabilities required in opportunistic networks.
   Available at `https://github.com/umr-ds/OPPLOAD`.

3. **Seamless Vertical Handover Demo-App**: This is a proof-of-concept implementation of the proposed seamless handover approach presented in *Learning Wi-Fi Connection Loss Predictions for Seamless Vertical Handovers Using Multipath TCP* [112].
   Available at `https://umr-ds.github.io/seamcon/`.

4. **DTN7-Go**: This is a reference implementation of the Bundle Protocol 7 [40] written in Go. The technical details were published in *DTN7: An Open-Source Disruption-Tolerant Networking Implementation of Bundle Protocol 7* [191].
   Available at `https://github.com/dtn7/dtn7-go`.

5. **BlueRa**: This is a cross-platform chat app that uses LoRa to realize long-range, infrastructure-free communication. The app was released as part of *LoRa-based Device-to-Device Smartphone Communication for Crisis Scenarios* [111].
   Available at `https://github.com/umr-ds/bluera`.

6. **ProgDTN**: This is a Go-based reference implementation of *ProgDTN: Programmable Disruption-tolerant Networking* [231]. It allows implementing routing protocols using Javascript for disruption-tolerant networks. It is integrated into DTN7-Go.
   Available at `https://github.com/umr-ds/dtn7-go/tree/ProgDTN`.

1.3 Organization of this Thesis

This thesis is organized as follows. In Chapter 2, fundamental concepts, technologies, and terms that are required throughout this thesis are explained.

The novel concept of situation-aware edge computing is introduced in Chapter 3. First the terms situation and situation-awareness are defined followed by the introduction of the area of edge computing and its sub-areas infrastructure edge, device edge and embedded edge. The final section formulates challenges and open questions for situation-aware edge computing and presents how situation-aware edge computing can be achieved.

In the following, the three areas of situation-aware infrastructure edge computing, situation-aware device edge computing and situation-aware embedded edge computing are discussed. Chapter 4 presents two approaches using situation-awareness in the area of infrastructure edge. The first part shows an approach where situation-awareness supports economic decisions about where to place services on infrastructure devices. The second part presents a method that uses situation-awareness to comprehend the service provider's situation with respect to where users use services.

Chapter 5 presents two approaches using situation-awareness in the device edge area: (i) an approach where mobile devices in opportunistic networks use their situation to decide on which mobile devices in the network a process should be offloaded to, and (ii) an approach that projects the mobile device's future situation regarding its connection state.

The concept of situation-aware embedded edge computing is discussed in Chapter 6. First, an innovative programming language is presented that follows the reactive programming paradigm and can be used to develop novel situation-aware applications executed on embedded devices. Second, an approach that uses a novel Complex Event Processing (CEP) engine to enable IoT devices to execute CEP queries to support the analysis of sensor information, such that situation-awareness can be achieved.

Chapter 7 discusses two case studies for situation-aware edge computing. The first case study shows how situation-aware edge computing can be used to support emergency response applications. The second case study presents how situation-awareness can be used to implement transitions in the device edge area.

Finally, Chapter 8 concludes this thesis and outlines areas for future work.

2
Fundamentals

In this chapter, fundamental concepts, technologies and terms will be introduced and explained that are required throughout this thesis. In Section 2.1, different aspects of communication networks will be introduced, whereas Section 2.2 will introduce two major network access technologies. Section 2.3 contains fundamentals regarding the concept of Internet of Things. Programming and data analysis concepts are introduced in Section 2.4. Finally, Section 2.6 establishes the concept of transitions.

2.1 Communication Networks

Communication networks are integral parts of decentralized edge computing. Thus, this section will first introduce the OSI reference model, followed by a discussion of disruption-tolerant networking and opportunistic networking.

2.1.1 OSI Reference Model

The International Organization for Standardization (ISO) proposed a seven-layer reference architecture of protocols, the Open Systems Interconnection (OSI). The main idea is to bundle services and protocols providing similar or even identical operations into layers and provide common interfaces between layers. This approach allows protocols within one layer to be modified without affecting neighboring layers [281].

Fig 2.1 shows the OSI model [281]. The figure depicts how data is transmitted from one end system to another with an intermediate node. Solid arrows represent dataflow within the system whereas dotted arrows represent logical network connections between the systems. The first layer called *physical layer* (often abbreviated as PHY) contains all aspects regarding how to use a medium like wires or radio frequencies, e.g., mechanical or electrical functionality to establish and maintain physical connections between network devices. The *data link layer* (abbreviated as MAC for medium access control) builds on top of the PHY layer. It controls medium access, i.e., reduces, detects and corrects errors and permits data to be sent to the PHY layer. Both the PHY and the MAC layer provide a point-to-point connection. Although the OSI model separates PHY and MAC layers, protocols used are usually combinations of both layers like IEEE 802.3 for Ethernet connections or IEEE 802.11 for Wi-Fi connections.

The upcoming layers provide end-to-end connections between devices possibly over multiple hops. The *network layer* provides data transfer between networks and from source to receiver

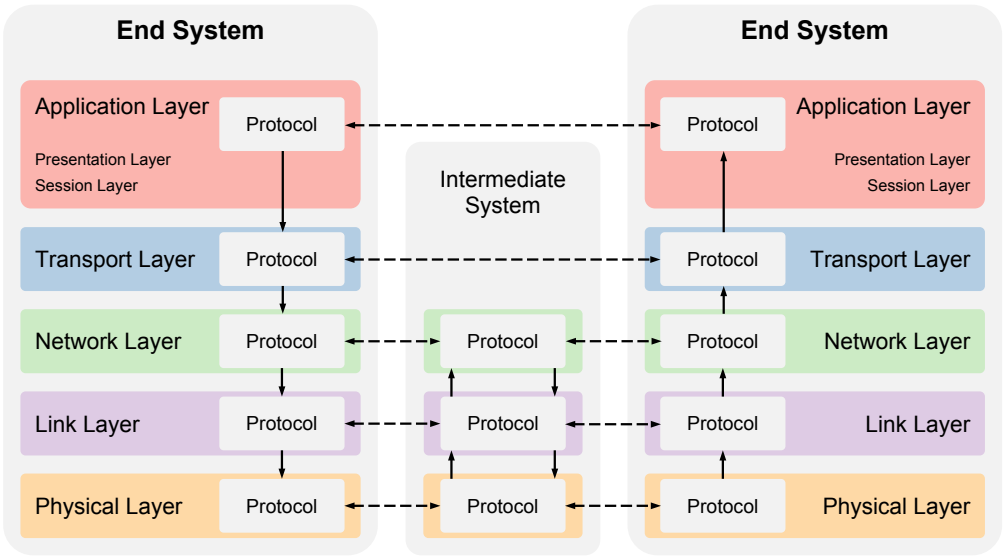


Figure 2.1: OSI model [281]

without knowing the route, where intermediate devices act as forwarders to allow data transfer over multiple hops. At this level, the Internet Protocol (IP) is the most widely used protocol. On top of the network layer the *transport layer* provides data transfer between services or sessions on devices without having to deal with the costs or requirements of the underlying layers. At this layer, two protocols are used ubiquitously: Transmission Control Protocol (TCP) for reliable and User Datagram Protocol (UDP) for unreliable data transfer, although other notable mentions exist like SCTP [237] and QUIC [138].

The first four layers were generally transport-oriented. The remaining three layers are application-oriented, since they to not handle data transport-specific operations, but application-specific functionality. Layer five is called *session layer* and is mainly concerned with functionality regarding data synchronization. The *presentation layer* is responsible for all matters related to the presentation of data. Both the session and the presentation layer are considered obsolete. There are no widely used protocols for both layers, although they exist, e.g., Remote Procedure Calls [193] for the session layer and Connectionless Presentation Protocol [118] for the presentation layer. The final layer is the *application layer*. It acts as the interface between the user-facing part and the application itself. This layer contains a plethora of protocols like the Hypertext Transfer Protocol (HTTP) for transferring and presenting hypertext, Domain Name System (DNS) for domain-name resolution or Message Queuing Telemetry Transport (MQTT), a lightweight publish-subscribe protocol for transmitting telemetry data especially on the Internet of Things area (see Section 2.3).

2.1.2 Disruption-tolerant Networks

Most networking protocols, especially those used on the Internet, are designed in a way that relies on stable point-to-point or end-to-end connections. If connections between devices are intermittent and error-prone these protocols do not work, making communications impossible. One example of such networks can be found in the area of interplanetary communications,

leading to the development of the InterPlaNetary Internet (IPN) in the 1970s [45]. Here, Cerf et al. [45] identified two major issues: (i) motion of and long distance between planets, resulting in long delays and intermittent connections, (ii) no fixed communications infrastructure, requiring constant route changes. Fall [77] noted other communications networks also have similar problems to interplanetary communications, for example in mobile networks using wireless communications, sensor networks or ad-hoc networks in emergency response applications. Thus, he generalized the concept if IPN, resulting in the concept of Delay-Tolerant Networks (DTN) [77]. Both efforts of Cerf et al. and Fall resulted in RFCs 4838 [46], 5050 [219] and the recently published RFC 9171 [40]. The term disruption-tolerant networking has evolved in the literature as a synonym for delay-tolerant networking.

In general, data transmission in DTN follows the store-carry-forward principle. Here every data unit, called bundle, is stored locally on reception and forwarded to other, neighboring nodes. In contrast to IP networks, where packets are only stored temporarily until the forwarding is finished (i.e., in a store-and-forward manner), DTN nodes store bundles persistently so that bundles can be forwarded multiple times, increasing chances that bundles receive their destination despite intermittent connectivity.

In DTN, nodes and endpoints are distinguished. Nodes exchange bundles according to the store-carry-forward principle. Bundles are addressed at endpoints, or, their characterizing Endpoint Identifier (EID), which might currently not exist in the network. Multiple nodes can have the same EID so that multicasts can be realized. A bundle addressed to an EID that is used by multiple nodes results in that bundle being transmitted to all these nodes [191].

Packets in a DTN consist of blocks to form logical units called bundles. Each bundle begins with one primary block, containing meta-information about the bundle: version, destination and source EIDs, timestamp, and size. Each bundle may have multiple canonical blocks containing block-specific characteristics. The payload of the bundle, i.e., the data to be transferred, is located in the payload block at the end of each bundle.

Each node must decide to which other node a bundle must be sent. To do so, a plethora of DTN routing algorithms was proposed in the literature. The most simple one is epidemic routing, where each node sends a bundle to all other adjacent nodes, resulting in flooding the network with the given bundle [248]. But also more sophisticated algorithms exist, most notably Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET) [148], Spray and Wait [232] and Delay-Tolerant Link State Routing (DTLSR) [67].

Bundles are exchanged over connections between nodes of different types and characteristics, connections are unidirectional or bidirectional, or vary in transmission speed and bandwidth. Depending on the connection technology used, more or less complex protocols are required for delivery, called Convergence Layer (CL) Protocols (CLP). There are two CLPs defined by the IETF DTN group to exchange bundles over a TCP connection, the bidirectional TCP Convergence Layer Protocol (TCPCL) [230] and the unidirectional Minimal TCP Convergence Layer Protocol (MTCP) [39]. In addition to transport layer CLs, there are approaches based on other technologies like Bluetooth, serial communication or E-Mail

Fig 2.2 shows the general procedure of sending bundles across devices in a DTN [40]. An application prepares a bundle and hands it over to the bundle protocol, i.e., the implementation containing the routing and management components. Here, the routing algorithm decides to which neighbors the bundle should be forwarded. For each forwarding, the bundle will be

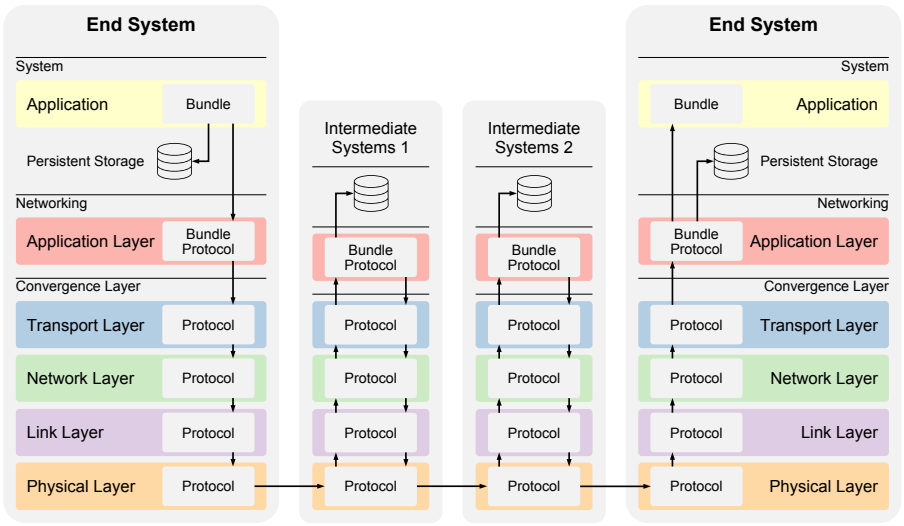


Figure 2.2: Data transfer from source to destination with intermediate nodes [40]

passed down the entire network stack, which is summarized as the CL. The bundle protocol layer is in Layer 7 of the OSI model. The CL contains all remaining layers, i.e., transport, network, MAC, and PHY. It might be possible that the CL also contains an application layer protocol like in the mentioned E-Mail-based CL. On reception of a bundle on the next node, the bundle is passed up through the entire network stack and persisted in the application layer by the bundle protocol. This entire process is repeated until the destination of the bundle is reached, where the bundle protocol will hand the bundle over to the application.

Opportunistic Networks

Opportunistic networks are specific cases in the domain of DTNs [33, 116, 245]. While DTN provides the conceptual and technical basis, opportunistic networks are concerned with the type of network and the distribution of data. They establish a mobile ad hoc network in which human-carried mobile devices connect directly via a short-range wireless technology like Wi-Fi or Bluetooth whenever they are in transmission range. In contrast to DTNs, which can include permanent communication infrastructures, opportunistic networks are designed to be infrastructure-free: nodes store data and carry it based on the underlying user mobility until a new communication opportunity to transmit the data emerges. The emphasis of opportunistic networks switches from user-centric to content-centric data distribution. This decreases network complexity, since selecting suitable intermediary nodes for information transmission is no longer a priority. Instead, data diffusion is dependent on human movement patterns as well as certain common content interests.

2.2 Radio Access Technologies

End user devices such as smartphones, computers or embedded systems can use various technologies to access the network. These access technologies can be broadly categorized into two groups: wired technologies such as fiber optics or Ethernet and wireless technologies like wireless LAN (WLAN) or 5G. Due to the nature of edge computing, wireless technologies play a far more important role, since devices used in such scenarios are mobile. Therefore, wired technologies will not be discussed any further. The remainder of this section will first introduce cellular networks used in public networks and WLAN mainly used in private networks.

2.2.1 Cellular Networks

Cellular networks is a collective term that combines many technologies and concepts. The historical development of the underlying technologies is described in generations, of which there are currently five [167]. The first generation (1G) deployment started in the 1980s, whereas the technological foundations date back to the 1970s. Since mobile devices were expensive at that time and only targeted limited markets, they never achieved commercial success. Further, 1G relied on analogous technology. During the 1990s, the second generation (2G) was introduced, which provided nearly the same services (mainly telephony), but used digital technologies as a basis. The main driving technology was the Global System for Mobile communications (GSM) protocol family, consisting of the General Packet Radio Service (GPRS) and Enhanced Data Rates for GSM Evolution (EDGE). This allowed better quality, lower cost, and range improvements. With the third generation cellular networks (3G), the main focus switched from telephony as the main service to data transmission, web-services and multimedia applications. With a data transmission rate of up to 14 Mbit/s, the throughput was increased about nine hundred times, mainly achieved by new protocols such as Universal Mobile Telecommunications System (UMTS) and High Speed Packet Access (HSPA). The main disadvantage of 3G was the design around so-called bearer services. 3G system designers defined service types such as speech, videotelephony or high data rate that influenced MAC layer properties to fulfil the requirements of these types. However, it turned out to be impossible to predict what type of service will be used, if new types appear that are not provided and how the requirements of the existing types will evolve. And in fact only the speech service and the high data rate service is used in modern 3G networks. This consequently mainly influenced the development of the fourth generation of cellular networks, 4G. The main protocol family of 4G networks is called Long Term Evolution (LTE). LTE is a data-only system achieving up to 100 MBit/s data throughput that was later improved to achieve 1 GBit/s using LTE-Advanced. The fifth generation (5G) promises data rates of up to 10 GBit/s with a capacity of up to one million devices per square kilometer (4G only achieves 10% of that capacity) and possible latencies of below 10 ms.

In real-world deployments, 1G is not used anymore, since 2G provided better service qualities in all areas. 2G, 3G, and 4G are currently used ubiquitously around the world. However, with 4G's large coverage, providers begin to rollback 3G, but not 2G. This leads to the separation of 4G used for all IP-based communication (including multimedia- and web-service but also speech in the form of Voice over IP, VoIP) and 2G as a fallback for telephony. However, with the increasing deployment of 5G mobile networks since 2019, 2G and 4G networks will no

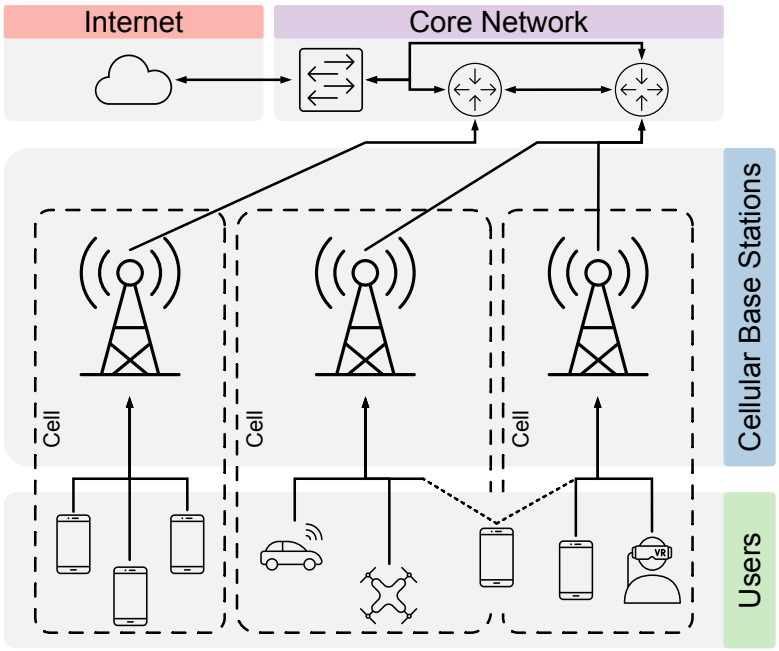


Figure 2.3: Schematic infrastructure and topology of cellular networks

longer be needed and shut down. Due to the significance of 5G as the new standard and the shutdown of 2G, 3G and 4G in the future, this thesis will only discuss 5G technologies.

Despite many technological differences of the five generations, they share the basic infrastructure and topology concepts [203], as shown in Figure 2.3. Cellular networks split their service area into discrete geographical units known as cells (hence their name). All wireless devices in a cell (denoted as users in Figure 2.3) connect with a cellular base station using radio waves via fixed antennas on frequency channels allotted by the base station. Furthermore, to be able to communicate with devices within the same network the base stations are connected with high-bandwidth backhaul links with each other to a core network, shown in purple in Figure 2.3. Base stations and a core network are usually owned by so-called Internet Service Providers (ISP), while many ISP exist. For connectivity with devices or services in networks of other ISPs, the base stations are also connected to the Internet via the core network. When a mobile device switches from one cell to another, it is seamlessly handed off to the current cell. However, due to the need of supporting higher bandwidths, lower latencies and providing network and Internet access to a steadily increasing number of devices, 5G introduces a plethora of new standards and technologies. Radio access is divided into two frequency bands, FR1 for frequencies below 6 GHz and FR2 for 24 - 54 GHz. FR1 uses channel bandwidths of 100 MHz, and many of the frequency bands are overlapping with 4G frequencies. FR2 is mainly used for high-bandwidth applications, which is achieved by the high frequencies used. However, high frequencies impose the problem that they are limited in distance and are not able to penetrate solid objects. Thus, the coverage and frequency usage in FR2 is highly application-specific and requires different deployments and cell sizes for the given application. There are four main cells used in 5G, that are summarized as small cells: (i) femto cells, (ii) pico cells, (iii) micro cells and (iv) metro cells. Each cell type is designed around a specific deployment environment. Femto cells are supposed to be used in homes or businesses with 4

to 16 users per cell with a range of tens of meters. Pico cells should be used in public areas like airports, train stations or shopping malls with up to 128 users per cell in a range of tens of meters. Micro cells are designed for urban areas to fill gaps of FR1 base stations, with up to 256 users per cell in a radius of few hundreds of meters. Finally, metro cells are used to provide additional capacity in urban areas for more than 250 users per cell in a range of hundreds of meters. Three additional technologies with less importance for this thesis are beamforming, Software-Defined Networking (SDN), and Network Function Virtualization (NFV). Beamforming involves shaping electromagnetic beams by arranging antennas in such a way as to make more efficient use of spectrum, increase range, and improve signal quality, resulting in significantly improved overall quality of service. SDN decouples the network control plane from the data plane, and NFV allows for instantiating logical network functions on top of a single shared physical network infrastructure. These two concepts allow optimizing network usage and a fine-grained control of networks, aiming to increase quality of service for users. Finally, 5G also adopts the concept of edge computing to reduce latencies of services, reduce traffic congestion and potentially reduce operational cost for providers of core networks and base stations as well as providers of services that are used by users.

2.2.2 Wireless LAN

WLAN describes a protocol family from the IEEE 802 standards, which are defined in IEEE 802.11 [60]. While the first standards and concepts were developed in the 1980s, WLAN did not gain acceptance until the early 2000s, when components and end user devices became affordable and the technology matured. Like all standards of the IEEE 802 family, IEEE 802.11 defines the lowest two layers of the OSI model, i.e. PHY and MAC layer. For IEEE 802.11, the MAC layer was largely taken over from IEEE 802.3, the standard for Ethernet, and only the interface for the PHY layer was adapted. The PHY layer, on the other hand, was developed completely independently, since it was the first wireless transmission medium in the IEEE 802 family. In addition, a goal of the IEEE 802.11 family was to transport IP packets in local area networks in an unlicensed frequency range so that they can be used without obstacles, especially in home or corporate networks, to provide simple, easy and wireless access to the Internet. Today, WLAN is the most widespread wireless Internet access technology. In addition to WLAN, the term Wi-Fi (wireless fidelity) is often used synonymously. However, it must be noted that these two terms describe the same thing, namely wireless access to the Internet using the IEEE 802.11 standards, but from different perspectives. WLAN describes the technical radio network defined by IEEE 802.11, while Wi-Fi describes the certification of end user devices according to the IEEE 802.11 standard. This circumstance is also evident in the fact that the naming of the standards or protocol collections differs between WLAN and Wi-Fi. At irregular intervals, the IEEE publishes new standards and amendments to them, which always have the name IEEE 802.11Y, where Y is a consecutive letter (IEEE 802.11a, IEEE 802.11b, ..., IEEE 802.11ac) [60]. Milestones are published as stand-alone standards (such as IEEE 802.11b, IEEE 802.11g, IEEE 802.11n, or IEEE 802.11ac), while standards in between represent extensions or amendments. The publications considered by IEEE as stand-alone standards are subsequently certified by the Wi-Fi Alliance under the name Wi-Fi X, where X is a version number. For example, Wi-Fi 4 corresponds to IEEE 802.11n or Wi-Fi 5 to IEEE 802.11ac. Throughout this thesis, however, the term Wi-Fi is used to indicate that only Wi-Fi certified, off-the-shelf devices are used.

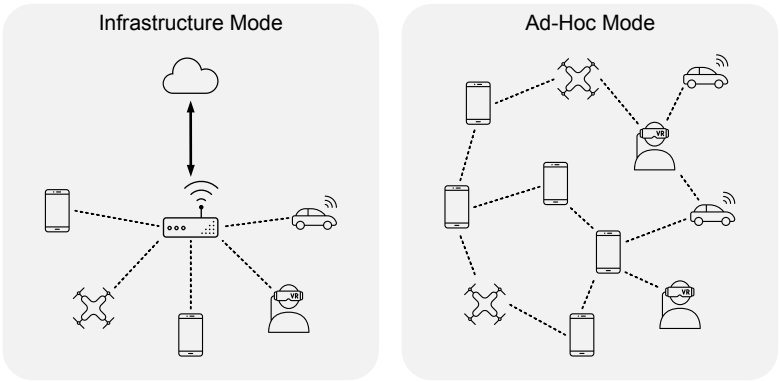


Figure 2.4: Two main Wi-Fi operation modes: Infrastructure mode and Ad-hoc mode

Since the existence of WLAN, the IEEE 802.11 standard has evolved considerably especially in the PHY layer, which has resulted in many improvements. While Wi-Fi 1, introduced in 1999, allowed a maximum data transmission rate of 11 MBit/s, Wi-Fi 4 (2003) already had 600 MBit/s and the latest standard Wi-Fi 6 (2020) up to 9.6 GBit/s. Wi-Fi uses two frequency bands, 2.4 GHz - 2.5 GHz (commonly referred to as 2.4 GHz) and 4.915 - 5.825 GHz (commonly referred to as 5 GHz). These bands are divided into channels, with the channels in the 2.4 GHz band being 5 MHz apart. Due to varying local regulations in different regions of the world, the channels in the 5 GHz band are not uniformly separated. Furthermore, depending on the frequency band, standards and regulations, channels can have a bandwidth of 20 MHz, 22 MHz and 40 MHz for the 2.4 GHz band and 20 MHz, 40 MHz, 80 MHz and 160 MHz for the 5 GHz band (although proprietary extensions exist, e.g., 66 MHz). To encode data into the frequencies, WLAN uses Orthogonal Frequency-Division Multiplexing (OFDM). OFDM divides the transmission channel (called carrier) into subchannels (called subcarriers) over which data can be transmitted independently of one another. In order to modulate data onto the subchannels, the information is encoded using the amplitude level. Various amplitude modulation methods can be used. If the transmission conditions are favorable, for example, the 64 Quadrature Amplitude Modulation (64QAM) method can be used. However, the worse the ambient conditions, the more robust amplitude modulation methods such as Binary Phase-Shift Keying (BPSK) can be used, depending on the standard. This can then achieve more robust communication over a longer range, but at a lower data rate.

Wi-Fi supports two different operation modes, as shown in Figure 2.4: (i) infrastructure mode and (ii) ad-hoc mode. For the infrastructure mode, a wireless Access Point (AP) coordinates all clients, called stations. Multiple stations use the same AP to connect to the network. In ad-hoc mode all devices are equal and are connected in a peer-to-peer manner. Because there is no central instance in an ad-hoc network, the stations must undertake the coordinating function themselves. Data packet forwarding between stations is not allowed for and is difficult to do in reality since no information is shared in ad-hoc mode that may provide individual stations with an overview of the network. To alleviate this issue, participating stations can be equipped with routing capabilities, allowing them to pass data between stations that are not in transmission range of one another, resulting in a mesh network. Besides these two main operation modes, Wi-Fi also supports hybrid modes in between like Tunneled Direct Link Setup (TDLS) or Wi-Fi Direct, where the AP serves as a negotiation service to help to establish peer-to-peer connections between stations.

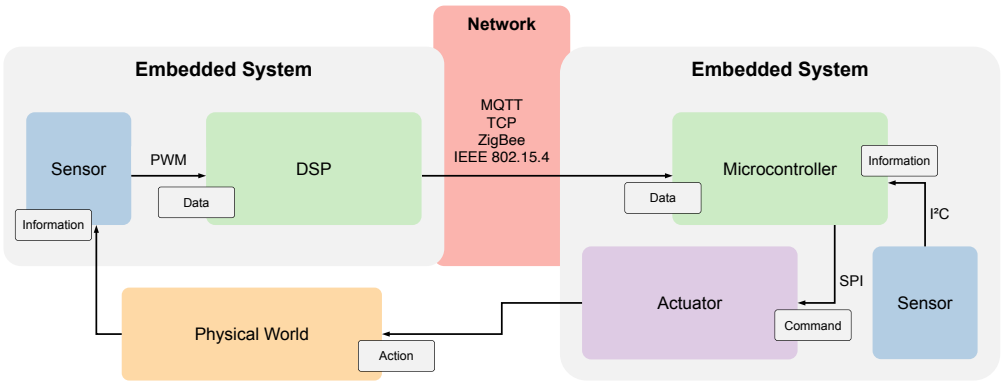


Figure 2.5: Structure of an IoT deployment [139]

On the MAC layer, IEEE 802.11 defines three types of datagrams called frames: (i) management frames used for managing the state of a connection like authentication, association to an AP or beacons to announce presence of APs, (ii) data frames, containing the actual data to be transmitted between Wi-Fi devices and (iii) control frames controlling data exchange by providing acknowledgements of received frames or indicating when stations are allowed to send. Each frame consists of a header containing information like frame type or source and destination address, and the payload in case of data frames. Medium access is controlled using Carrier-Sense Multiple Access with Collision Detection (CSMA/CD). When a station is ready to send data, it sends a Ready to Send (RTS) frame to the AP or other station in case of ad-hoc mode. The AP or station handles this frame accordingly either by ignoring it when another station is sending or responding with a Clear to Send (CTS) frame. Collision detection is implemented by using acknowledgement frames. Every transmitted frame has to be acknowledged by the receiver. If the sender does not receive the acknowledgement, a collision is assumed and retries sending the data after a random back-off.

2.3 Internet of Things

Internet of Things (IoT) is a term name describing technologies that allow physical and virtual items to be networked and communicate with one another via information and communication techniques. To enable the IoT, two areas are usually required: embedded systems for computing and executing functions, and networking and communication between these systems. Figure 2.5 [139] shows the structure of an IoT deployment containing of multiple embedded systems and their communication network.

2.3.1 Embedded Systems

An embedded system is a computer that is embedded in a technical setting and performs monitoring, control, or regulation activities, or is in charge of some type of data or signal processing, such as encoding, decoding, or filtering [139]. Embedded systems are frequently tailored to a specific goal. Embedded systems are frequently subject to severely constrained conditions, such as low cost, limited space, energy, and memory usage, which necessitates the

employment of specialized hardware or appropriate designs. In general, only limited resources exist. A ROM or flash chip substitutes mechanical memory components such as hard drives, and power-saving CPUs operate without fans since moving parts create wear and tear and make fall-outs more likely. Furthermore, the software in an embedded system, referred to as firmware, is typically required to satisfy real-time needs. It is often stored on a ROM or flash memory. All of these criteria necessitate concepts and technology that enable a suitable platform for their duties to be realized despite these constraints.

Processors used in embedded systems can be categorized into two categories: (i) microcontrollers and (ii) Digital Signal Processors (DSPs). A microcontroller is a small computer that combines a CPU with peripheral devices such as memory and I/O devices on a single integrated circuit [139]. They frequently employ 8-bit architectures and are well suited to applications that require low memory and basic functionalities. They may use a minimal amount of energy and often have dedicated features like as a so-called sleep mode, in which components of the CPU are turned off, reducing power consumption to nanowatts and allowing them to run for several years on batteries. However, separating microcontrollers from general-purpose processors is not as simple as it may appear, since several CPUs (e.g., the Intel Atom family or AMD Geode) fulfill some above-mentioned features but are deployed in devices as general-purpose CPUs. Filtering, system identification, frequency analysis, machine learning, and feature extraction are just a few of the advanced mathematical processes that many embedded programs execute on data. DSPs are processors that are developed specifically to serve such numerically intensive signal processing applications and are often microcontrollers themselves.

Besides the communication between different embedded devices themselves, CPUs also have to be able to interact and communicate with periphery or sensors [139]. Therefore, embedded systems often include numerous I/O hardware components and interfaces. These include hardware for Pulse Width Modulation (PWM), where a variable amount of data is delivered to periphery, or General-Purpose Digital I/O (GPIO) to allow digital communication protocols such as Inter-Integrated Circuit (I^2C) or Serial Peripheral Interface (SPI). But also serial interfaces like Universal Asynchronous Receiver/Transmitter (UART), or buses like Peripheral Component Interface (PCI) are possible.

Finally, embedded systems are usually programmed using imperative and low-level programming languages, predominantly C. Programs are then either executed on bare metal, i.e., without any operating system or by using simple operating systems. Often, embedded systems do not support concurrent or parallel execution of programs but only sequential execution. However, due to the popularity of IoT, more modern operating systems provide interrupt-based concurrency, support multitasking, and parallel execution if multiple CPU cores are available, e.g., the ESP platform by Espressif Systems.

2.3.2 Networking and Communication

The challenges of embedded systems are mainly resource constraints [120]. This makes it necessary to design new network protocols for many applications. In addition, the many possible uses and countless applications of IoT make it impossible to develop a single protocol or only one protocol family. Instead, a number of different protocols have been developed at almost all levels of the OSI model [120].

Similar to conventional networks, PHY and MAC layers are often specified jointly. One of such protocols is Z-Wave, a low-power protocol to support point-to-point communication in a range of about 30 meters. It is designed for scenarios like home automation, where one device acts as the controller that controls multiple agents like light switches, or fire detectors. By implementing a collision avoidance scheme and acknowledgements, it is considered a reliable protocol. Bluetooth Low Energy (BLE) is often used in IoT applications for pairing of devices. IEEE 802.15.4 is designed to support different application scenarios by varying the transfer rate (between 40 kbit/s and 250 kbit/s) with the trade-off of increased power consumption at higher transfer rates.

On top of PHY and MAC layers, there are also a number of network layer protocols designed specifically for the needs of IoT, although the IP protocol is also used widely. ZigBee, which is built on top of IEEE 802.15.4, is designed to support many devices (up to 65,000), being less expensive than BLE and Z-Wave and to support a decentralized topology. To support an IP-based alternative with IEEE 802.15.4 as its foundation, IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) offers similar benefits as ZigBee, with the main difference of using IPv6 for routing data though the network. TCP and UDP are not only used almost without exception in conventional networks, but also in the area of IoT, since there are no alternatives available that are used in real-world deployments.

For application layers, one widely used protocol is the Message Queue Telemetry Transport (MQTT) protocol. It relies on TCP as the transport protocol and transfers data in a publish-subscribe manner, where publishers send data in so-called topics to a broker, where subscribers can read the data. It also supports three quality of service classes: (i) at most once, where data may or may not be delivered, (ii) at least once, where data is re-transmitted until the broker acknowledged its delivery and (iii) exactly once, where delivery is assured. Built on top of IEEE 802.15.4, UDP and HTTP, the Constrained Application Protocol (CoAP) is designed for a point-to-point state transfer instead of a many-to-many data transfer scheme as implemented by MQTT.

One challenge in IoT networks is that devices are often dynamically added to and removed from the network. This imposes the need for service discovery, so that newly added devices get to know other nodes and services and to inform the network of removed devices. Here, mainly two protocols are used, which work similar, since both rely on the Domain Name System (DNS) protocol Multicast DNS (mDNS) and DNS Service Discovery (DNS-SD). They support automatic configuration of devices, they do not require additional infrastructure, they are fault-tolerant, and support service discovery by relying on the multicast abilities of IP networks. These protocols are just examples to illustrate protocols designed specifically for IoT and embedded systems.

2.4 Programming Paradigms and Data Analysis Concepts

Due to the steady increase of devices on the Internet and the resulting ever-growing amount of data, it is necessary to develop programming paradigms and data analysis concepts that are easy to use and support different execution environments, from servers to embedded systems. Three of these approaches, Complex Event Processing, Reactive Programming, and Remote Procedure Calls, are presented in the following.

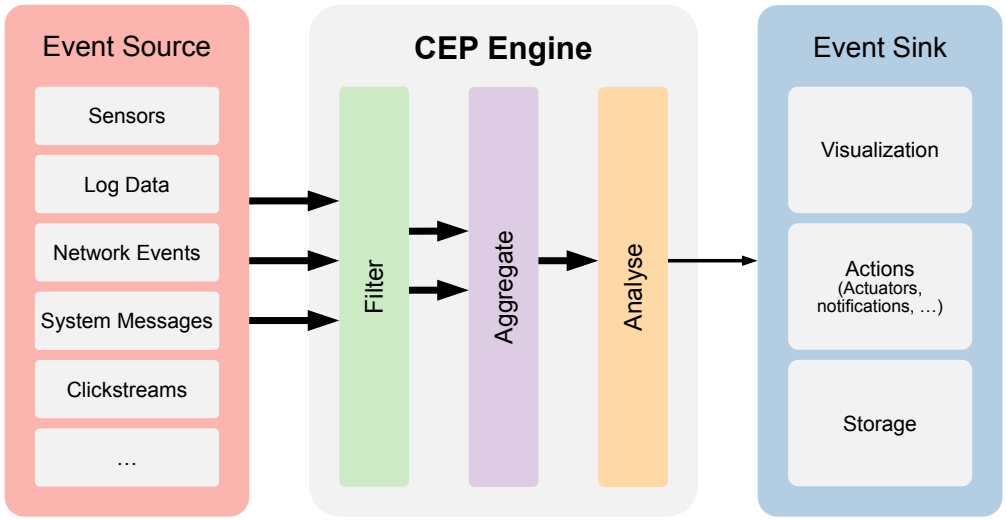


Figure 2.6: Structure of complex event processing pipeline

2.4.1 Complex Event Processing

The term Complex Event Processing (CEP) was introduced in the late 1990s [153]. Key concepts include how to specify event patterns and the elements of programming languages required to do so, how to build efficient event pattern-detection engines, strategies for using event patterns in event processing, and how to define and use hierarchical abstraction in processing multiple levels of events for various applications. CEP systems frequently deal with many event streams from several sources at the same time, as well as different sorts of events with event rates between a few events per hour to thousands of events per second.

Changes of values or the input of new values are called events [153]. One event is a single change, while usually events occur as streams of changes at discrete points in time. For example, the constant typing on a keyboard results in a stream of changes or a sensor producing data results in one event for every new sensor value which forms an event stream. In Figure 2.6, events are produced on the left side by sensors that are then passed to the CEP engine, i.e., the building block where the events are processed. However, the results of processing events can produce new events that are fed back into the CEP engine.

The initial stage of an event engine is to use simple criteria to eliminate events that do not contribute to the processing's goals, a process known as filtering, where the criteria are called filters. After the filter, more advanced methods are used to analyze the events, such as applying statistical algorithms, grouping events based on similarities, or using aggregating methods over time frames. These actions will result in the creation of new events that contain the summary data. Furthermore, event-abstraction approaches enable development of systems capable of analyzing large numbers of low-level events and generating high-level events that abstract the knowledge contained in those events. Thus, they are fewer but more significant since they provide views of the information contained in the event sources.

All events in CEP are immutable, making it impossible to change or delete an event [153]. If an event contributes to the processing in any manner, the original event must always be retrievable if necessary. Copies with changes to data or other attributes can be made.

Prioritizing events as they occur is also a crucial first step. High-priority occurrences are those that necessitate prompt action. In reality, prioritizing can be performed before or after filtering, or the two operations might be performed in a different order on the input-event streams. Computations make use of the data included in events. This step could include simultaneous computations on the contents of the events. At this step, CEP allows to use any algorithm for further computation. In reality, a series of algorithms might be run at the same time, with the new events being added to the event stream for the next steps. Pattern detection is another type of analysis. An event pattern is a template that can match several sets of multiple events, or many single events in the case of single event patterns. It may necessitate the presence of specific data in events, or the events in a matching set to be related in various ways, e.g., some events may be required to be causes of other events in the set, or to occur within a specific time window. Some sorts of events may be required to be absent in an event pattern. Exceptions are errors and anomalies that are detected as a result of event processing. The existence or absence of events and patterns of events are used to detect exceptions in CEP. Exceptions are dealt with by producing new events that represent the exact problem that occurred. When a pattern of events is recognized, the processing may create new events using properties and data from the set of events that matched the pattern. Events are frequently categorized into layers, where some events are low-level, while others are high-level. Higher-level events are made up of groups of lower-level occurrences. Event hierarchies have several aims, including (i) focusing information for specific users and (ii) reducing the quantity of events that must be processed at higher levels. At the end of the CEP engine, some sort of action is triggered, or the gathered information is used otherwise (e.g., visualized in a dashboard). This process is shown in Figure 2.6.

From a technology standpoint, CEP systems employ domain-specific programming and query languages to describe patterns and represent data processing stages. Stream SQL (SSQL) and SQL variations like Continuous Query Language (CQL) are the most common of these, with modifications for defining events and patterns. One reason for SQL variations' prominence as the preferred event pattern specification language is its simplicity of implementation, intuitive handling and early CEP frameworks being based on database research. Languages that define event patterns using regular expressions are also used, for example, as extensions to existing general-purpose programming languages.

2.4.2 Reactive Programming

Reactive programming is a programming paradigm that was established by Elliott et al. in their implementation of Fran [74], and is built around continuous event streams and propagation of change caused by events and their evaluation [19]. Applications that operate on continuous data streams, e.g., in the area of IoT, where sensors produce continuously data that has to be evaluated and actions have to be performed like triggering actuators, are difficult to program using conventional control flow drive programming concept due to the unpredictability of event arrival. To overcome this shortcoming of control flow driven languages, so-called callbacks are used, where logically independent chunks of code asynchronously operate on shared data. However, coordinating callbacks is a tedious task, since they have to utilize side effects to change an application's state. This leads to the problem that such event handling logic contributes to nearly a half of the bugs reported, thus, this design was coined callback hell [19]. In reactive programming, however, the control flow is inverted, i.e., the control flow

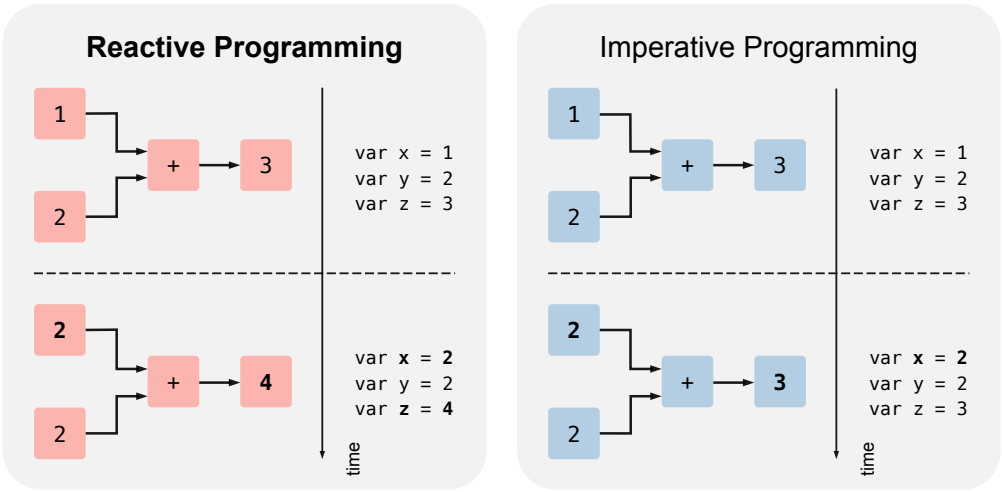


Figure 2.7: Comparison of reactive programming ([19]) and imperative programming

of a program is driven by events and not by imperative program statements. By providing abstractions (i) to express programs as reactions to external events (ii) having the language automatically manage temporal, (iii) data and (iv) computation dependencies of events, reactive programming yields the advantage that developers do not have to worry about the order of events and computation dependencies.

As a rough overview, using reactive programming, programmers implement what to do and the language automatically manages when to do it, based on the input of events. To illustrate this concept, consider the following example shown in Figure 2.7 [19]. The variables `var x = 1` and `var y = 2` are assigned values and variable `var z = x + y` is the combination, in this case addition, of these variables `var x` and `var y`. In most programming paradigms, such as imperative or object-oriented programming, the variable `var z` is assigned the value 3 that is, the evaluation of the addition of x and y, i.e., `var z = x + y = 3`. If now one of the two or both values of `var x` and `var y` change, the value of `var z` remains with the mentioned programming paradigms on `var z = 3`. To update the value of `var z` the addition would have to be executed again. In reactive programming, on the other hand, the value of `var z` changes as soon as a change is made to `var x` or `var y` or both. Thus, the programmer does not have to worry about it. In this example, it seems trivial. In complex data processing tasks, however, the programmer would have to constantly make sure that all references between data and the functions are correct, up-to-date, and valid. With reactive programming, all these steps happen at the language level.

The term event in reactive programming follows the same definition as in the CEP domain. However, in reactive programming events are first-class values and are composable. Most languages provide primitive combinators to arbitrarily combine events or filter a sequence of events. To be able to react to a new event, the change has to be propagated, i.e., dependent computations need to be notified so that they can be re-evaluated. To do so, a dependency graph of values and computations is built that is used to organize the propagation. The propagation can be performed in two ways, either push-based or pull-based. In the former case, the source of the event pushed the event to all dependent consumers like functions that operate on the event. This results in fast recomputations because consumers perform

their computation as soon as new data is available, but at the cost of potentially inefficient programs as recomputations may happen often. In pull-based propagation, consumers pull events from sources on demand, adding the flexibility of only performing recomputations when they are needed instead of continuously. However, the pull-based method may result in unbearable latencies or so-called leaks in time and space: "a time leak in a real-time system occurs whenever a time-dependent computation falls behind the current time because its value or effect is not needed yet, but then requires catching up at a later point in time. This catching up can take an arbitrarily long time (time leak) and may or may not consume space as well (space leak)" [19]. One special pitfall of push-based propagation models are glitches, i.e., update inconsistencies during propagation. When a computation is run before all of its dependent expressions have been evaluated, new values may be mixed in with old ones, resulting in a glitch. Consider the following illustrative example: `var x = 1`, `var y = x * 1` and `var z = x + y`. The value of the variable `var y` should always be the same as that of `var x`, and the value of `var z` should always be double that of `var x` in this example. When `var x = 1`, `var y = 1`, and `var z = 2` at first, If the value of `var x` is changed to 2, for example, the value of `var y` will also change to 2, while the value of `var z` will remain at 4. Setting `var x = 2` may, however, lead the expression `var z = x + y` to be recalculated before the expression `vary = x * 1`. As a result, the value of `var z` will be 3 for a little period of time, which is wrong. Eventually, the phrase `var y = x * 1` will be recalculated to give a new value to `var y`, and `var z`'s value will be recalculated to reflect the right value of 4. Glitches can be avoided by sorting the dependency graph topologically.

In the recent past, languages like Fran or REScala [206] were proposed. But also outside the academia, FRP languages are used, for example Svelte.js[1] and Elm[2] for web-based frontends or ReactiveX[3], which provides a reactive API for general-purpose languages.

2.4.3 Remote Procedure Calls

A remote procedure call (RPC) is a call of a procedure, subroutine or function to run in another address space or on a different computer on the network, as if it were a regular local call. Programmers do not explicitly code information for RPCs, instead they write the same code regardless of whether the function is local or remote, implying location transparency. It is a type of client-server communication in which the function caller is the client and the procedure executor is the server.

Figure 2.8 depicts the steps required to perform an RPC [193]. The client first calls a local stub for the process, handing it the required arguments. The stub hides the fact that the procedure was called remotely. It converts the arguments to be part of the RPC protocol. The RPC protocol implementation then transmits the call to the server's RPC protocol implementation, which passes the call to the server stub. The RPC is translated into a local procedure call by the server stub, which is then executed locally on the server. After completion, the server sends a reply message, which is sent the back to the client. The received result is passed up to the client stub, which then returns it to the calling program.

[1] `https://svelte.dev`
[2] `https://elm-lang.org`
[3] `https://reactivex.io`

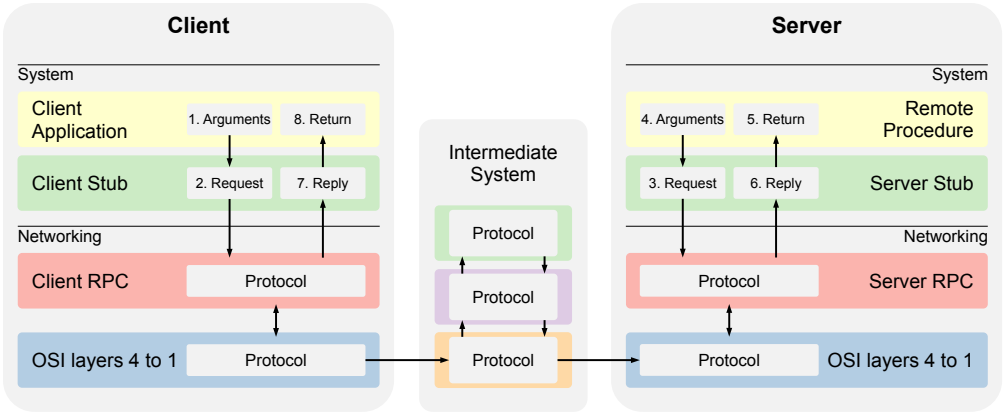
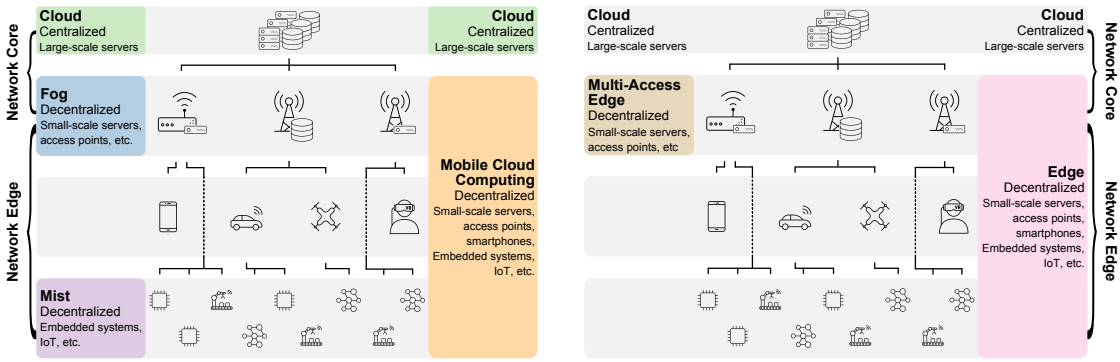


Figure 2.8: Schematic RPC embedded in the OSI model based on [193]

The stubs are in charge of two things. First, the client stub serves as a client-side proxy for the remote server operation, whereas the server stub serves as a server-side proxy for the calling client code. By hiding the "remoteness" of the code on the other side, the stubs effectively simulate a local function. Second, the stubs serialize local data so that it may be transferred via the network protocols in use. On the receiver side, the stub deserializes the network data so that it may be processed locally. Additionally, the stub encodes the serialized data into packets that the RPC protocol can process.

The RPC protocol has three main functions: (i) make RPC uniquely identifiable, (ii) correctly match results to corresponding calls, and (iii) reliably deliver messages between client and server. The concept of namespaces is used to implement the first function [193]. Namespaces ensure that names in a given environment are unique and, in the case of RPCs, procedures can be identified uniquely. Namespaces are either flat or hierarchical. An example for a flat namespace is the IP protocol. Addresses in the IP namespace are four bytes in size and globally unique (with some exceptions). The problem with flat namespaces, however, is that there has to be some central entity coordinating address or name assignment to avoid conflicts. In contrast, hierarchical namespaces provide the ability to assign names that only require uniqueness in the lowest level of the hierarchy. To give an example, a file path is structured as a hierarchical namespace. For matching calls and results, i.e., the second function of the RPC protocol, usually some sort of ID is used, e.g., a hash. Sequence numbers are also used. However, sequence numbers bear the problem that they are not unique. A client could (e.g., due to a reboot) issue the same sequence number twice. A server could then identify the particular call as a duplicate and reply with the wrong result. Thus, uniqueness for call IDs must be ensured. To reliably deliver messages between client and server, most RPC implementations rely on the TCP/IP protocol stack. However, the RPC protocol adds the at-most-once semantics on top. This means that a call must be delivered at most once to avoid unnecessarily executing the same call more than once. Taking the above discussed namespaces and call IDs into account, the server usually preserves the state of which calls were already executed. In case of a duplicate call, the server may either ignore it or return the cached result from the previous call.

SunRPC (or ONC RPC, Open Network Computing RPC, which is the standard developed by the IETF based on SunRPC) is the implementation used for the Network File System



(a) Cloud, fog, mist and mobile cloud computing **(b)** Multi-access edge and edge computing

Figure 2.9: Comparison of different computing paradigms

(NFS), an application layer protocol to support file systems in local networks. SunRPC allows implementing Unix system calls to be executed on remote computers. SunRPC uses a three-layer namespace, where the first layer is the computer's IP address, the second layer is a program identifier to identify the remote program on the remote computer and a procedure number to identify the remote procedure within the remote program. For reliable delivery, SunRPC implements a protocol on top of UDP that uses acknowledgements. Another well known RPC implementation is gRPC, which relies on HTTP, TLS and TCP for reliable data transmission and computer and program identification and the HTTP URI scheme for hierarchical procedure identification.

2.5 Computing Paradigms

Computing, i.e., to process, to structure, to find and to gather various kinds of information [220] can be divided into multiple categories, e.g., based on capabilities like available resources or on locations like centralized clouds or decentralized mobile computing. The latter categorization, i.e., where the computation is executed, relied for a long time on centralized mainframes that either executed the processes in batches or in a time-sharing manner. This approach relied on the capabilities of the mainframe vendors like IBM. With the advent of cloud computing, the centralized resources were democratized, enabling engineers, developers but also researchers to have access to nearly unlimited computing resources.

However, despite almost infinite resources, cloud computing suffers from different limitations. For example, in latency-sensitive applications like mobile augmented reality games, the time from initializing a process until receiving the result may not be fast enough, introducing unwanted or even unbearable latencies. Thus, in the recent past, new computing paradigms emerged to cope with the shortcomings of centralized cloud computing approaches. The new paradigms aim to bring the computation closer to the end user, i.e., to execute processes not in the central cloud but on other instances like access points or base stations or even smartphones and embedded devices, called fog computing, mist computing, mobile cloud computing, multi-access edge (or mobile edge) computing and edge computing. This section introduces there paradigms and presents differences between these approaches.

2.5.1 Cloud Computing

The origins of cloud computing can be traced back to the 1960s, when McCarthy and Parkhill envisioned computing systems as public utilities [122]. Nowadays, cloud computing has become the de facto industry norm, with all major Internet companies running their own platforms and making them available to others. The primary notion behind cloud computing is that users access their data and associated computations via the Internet on remote servers that are large-scale, redundant, and scalable. Scalability is defined as the ability to handle virtually unlimited amounts of work and to add or remove resources on short notice, also known as elasticity. Scalability is achieved through software-controlled deployment and removal of computing instances such as virtual servers, as well as the partitioning of data, configuration, and network- and computing resources so that clients are isolated from each other. The cloud infrastructure is maintained by the cloud computing provider, who is paid on a pay-per-use basis by the clients, which, when combined with the above-mentioned characteristics, allows clients to reduce their operating costs by eliminating the need to maintain and secure their own infrastructure. In addition, service-oriented architecture, which consists of a collection of loosely connected, platform-independent, and distributed services accessed via Internet protocols, plays an essential role. Quality-of-Service (QoS) metrics such as availability, throughput, latency, and others are defined in service level agreements, which are contracts between a cloud provider and a client that guarantee a specific level of QoS and define penalties for noncompliance. Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) are the three main service models that have emerged. The IaaS service paradigm connects computing resources such as processing, storage, and networking. Virtualized resources, or logical entities that are abstracted from the underlying physical components, are controlled by the client. The PaaS service model provides a programming and execution environment in which a client can develop and run software using a provider-defined programming language. Software applications and services that run in a cloud computing infrastructure, on top of PaaS or IaaS resources, and directly address a user are referred to as SaaS services. Full-fledged apps are often accessed universally from a thin client via a web browser. Finally, in cloud computing settings, four administrative layers are common: public, private, community, and hybrid clouds. The public cloud model allows service providers and consumers who are not affiliated with the same company to access services. Private clouds, in contrast to public clouds, provide services to users within a single enterprise. Nonetheless, cloud infrastructure software is employed to give members of this organization clean service interfaces. In settings where both public and private cloud services are utilized, a community cloud provides services for exclusive usage by a specific community with shared concerns, whereas hybrid clouds are used in scenarios where both public and private cloud services are used.

2.5.2 Fog Computing and Mist Computing

According to the OpenFog Consortium, the cloud computing paradigm is insufficient to solve problems arising from the recent IoT developments in 5G [183]. Fog computing is defined by bringing computing services closer to the user and the devices that create the data needed for computations [119, 183]. It is a horizontal architecture that distributes processing, storage, control, and networking resources and services anywhere along the continuum from

the cloud to end-devices, allowing for faster decision-making. Fog computing addresses a subset of business challenges that cannot be solved using typical cloud-based architectures or intelligent edge devices alone. It consists of several layers of fog nodes that are the architecture's main components. They can be arranged in vertical or horizontal clusters to support isolation or federation. Nodes at the edge of the network are often focused on data generated at the edge of the network, e.g., from mobile devices, sensors, or actuators. Next-tier nodes filter and transform data while also performing latency-critical analytics. Higher-tier nodes, located near fog nodes on cloud instances, are often utilized to convert data into knowledge. Fog computing, in contrast to cloud computing, provides minimal latency due to its proximity to end-devices and allows for unobtrusive dataflow for specific applications thanks to interoperating fog nodes and federated services. Furthermore, rather than batch processing, fog computing relies on real-time interactions. Microservices, which enable more fine-grained application structures aimed at improving modularity and manageability, are mostly utilized as a specialization of SOAs in addition to cloud computing. Additionally, similar to cloud computing, NIST defines three architectural service models for fog nodes (IaaS, PaaS, and SaaS) and four deployment options (public, private, community, and hybrid). Furthermore, NIST [119] defines mist computing as an additional layer that makes use of more specialized and dedicated nodes at the network's edge that have limited processing resources. Finally, the cloud-to-end-devices continuum is built in a hierarchical manner. Cloud services are located in centralized data centers at the network's core. Figure 2.9a shows the architecture of fog computing and mist computing. Fog nodes are cloud-connected server clusters such as cloudlets/micro-datacenters or edge devices such as wireless access points/routers that are near to the network's edge. Finally, mist nodes are IoT devices such as sensors/actuators or edge devices such as switches/routers that execute lightweight operations directly on sensors/actuators or other network fabric components, especially in industrial applications. The bottom layer of the hierarchical cloud - end device continuum is made up of mist nodes.

2.5.3 Mobile Cloud Computing

Fog computing brings the cloud to the edge of the network. Nonetheless, Mobile Cloud Computing (MCC), which, as shown in Figure 2.9a, integrates mobile resources into the cloud computing paradigm by exploiting scalable cloud resources for mobile devices, arose in parallel to fog computing with similar goals, but a different strategy [2]. However, MCC flips the perspective. Fog and mist computing give computing resources close to end users that service providers can leverage to deploy their services. In MCC, end users or their devices, rather than service providers, offload local computation to nearby computing resources, e.g., to preserve local resources like power or data storage, or to take advantage of special functions that can make execution faster or more accurate. MCC, like fog or mist computing, is viewed as an extension of cloud computing rather than a replacement. Distance and mobility are two properties that can be used to categorize resources in MCC. Cloud computing provides large-scale pools of centralized computer resources over long distances. Proximate computing entities, such as mini cloud-datacenters near the edge of the network, supplement these distant computing units. Both cloud resources and resources at the edge of the network are immobile. To provide additional offloading resources in a mobile way, mobile computing entities in the form of ad-hoc connected groups of mobile devices are deployed. MCC's resources, like those in fog and mist computing, can be organized hierarchically. Mobile phones can offload

computational and storage intensive tasks to immobile, distant entities represented as public cloud resources, forming the highest tier. The computation can also be offloaded to immobile entities in the second layer of the hierarchy closer to the offloading device, depending on the requirements. Furthermore, in the bottom tier of the hierarchy, services and network applications can be offloaded to nearby MCC nodes that are made up of mobile, ad-hoc connected devices. The higher the tier, the more resources for computation and storage are available, but latency increases and mobility support decreases.

2.5.4 Multi-Access Edge Computing

Both fog (and mist) computing and MCC represent an extension of the cloud. In both paradigms, different classes of devices with different capabilities and degrees of mobility provide decentralized resources in addition to the cloud. This changes with multi-access edge computing (MEC, formerly mobile edge computing). The goal of MEC is also the reduction of latency and the ability to handle future ever-growing data volumes by placing computing resources close to end users [1, 188], visualized in Figure 2.9b. However, it is not intended to extend the cloud but rather to provide an additional option to the cloud. This does not mean that the cloud will be replaced entirely, but that certain applications that rely on the benefits of MEC will run on the MEC infrastructure. Nevertheless, the cloud can be used as a fallback if no MEC infrastructure is available or the capacities are not sufficient. MEC provides storage and computation resources at the edge of the network, i.e., at base stations or small-scale servers called *cloudlets*. Application providers can offload processes to compute nodes that are usually located in a one-hop distance to their users. Finally, ETSI [188] defined requirements that have to be met in order to be able to fulfil the promises of MEC. These requirements include functional (e.g., requests have to be finished within a defined time frame) and non-functional aspects (e.g., the ability of transparently relocating the execution between the MEC and the central cloud). The design of the MEC concept is mainly driven by the ETSI and telecommunication providers, especially for 5G networks.

2.5.5 Edge Computing

Edge computing combines parts of all presented paradigms [129, 130]. Figure 2.9b shows edge computing in the networking and compute stack. Similar to MEC but unlike fog computing and MCC, edge computing is not designed as an extension of but an alternative to the cloud to improve the quality of certain applications or to enable novel applications. In contrast to MEC, edge computing is not limited to the infrastructure of telecommunications companies at the edge of the network, such as base stations or access points, but, like fog computing and MCC, also integrates end devices and embedded systems as compute nodes. Finally, edge computing not only places services of service providers on decentralized infrastructure that can be used by users, as is the case with Fog computing, but end devices can also offload local functionality to other edge devices, similar to MCC. Edge computing thus represents the most universal approach. Even if not all aspects of the other paradigms are implemented, which means that all the approaches presented, i.e., Fog Computing, MCC, MEC, and Edge Computing, are widespread used.

2.6 Transitions

The concept of transitions originated from the area of communication networks and was developed by the Collaborative Research Center *Multi-Mechanism Adaption for the Future Internet (MAKI)*, funded by the German National Science Foundation [10, 90]. In the recent past, however, the concept was also applied to computation transitions [105]. The goal of transitions is to maintain or even increase the quality of a service, network connection or other parts of a system by replacing one mechanism with another and thus in response or even proactively to changing environmental conditions. A simple example would be a handover between Wi-Fi and LTE without interrupting connections to maintain a high-quality Internet connection. This section will first introduce the ideas of mechanisms and multi-mechanisms and then discuss mechanism transitions.

2.6.1 Mechanisms and Mechanism Transitions

Transition, regardless whether in networks or computations, rely on the notion of mechanisms. A mechanism is a system's conceptual element that is tied to its manifestation as a system of collaborating functional elements. Thereby, mechanisms can be found in many areas of a system, e.g., a particular layer of the OSI model or specific functionality that is only provided by certain protocols of a specific layer (e.g., reliable data transfer in the transport layer as supported by TCP). Furthermore, a mechanism can be a functional part of a network protocol like the routing algorithm of the network layer. And mechanisms can also be functions or procedures that are executed remotely.

"A mechanism transition is the functional replacement of a source mechanism by a functionally similar or equivalent target mechanism in a running system" [90]. This implies that the mechanisms used to transition between are designed in a way that makes them exchangeable. An example would be TCP/TLS and QUIC as two mechanisms. Both are transport layer protocols with respect to the OSI layer and both provide the same basic functionality, i.e., reliable and secure data transfer between two processes running on different networked computers. Exchanging TCP with QUIC (or vice versa) due to changes in the environment (e.g., changed requirements with respect to bandwidth and latency, as QUIC achieves better latency and TCP better bandwidth utilization) in a running system, i.e., at runtime without stopping the system or processes on the system, is called a transition.

2.6.2 Multi-Mechanisms

A multi-mechanism contains multiple mechanisms that are similar but achieve their best results in different environmental conditions, and the transitions between them. Figure 2.10 [105] shows three arbitrary OSI layers, in this example the MAC, network, and layer. Each layer shows two different mechanisms with transitions between them. On the PHY layer, the mechanisms are the BPSK and 64QAM modulation algorithms, on the network layer IPv4 and IPv6, and on the transport layer the mechanisms are TCP and QUIC. In either example, the mechanism provide the same functionality, i.e., reliable data transmission for the transport layer, network routing for the network layer, and data modulation for the PHY layer, but for different scenarios and environmental conditions.

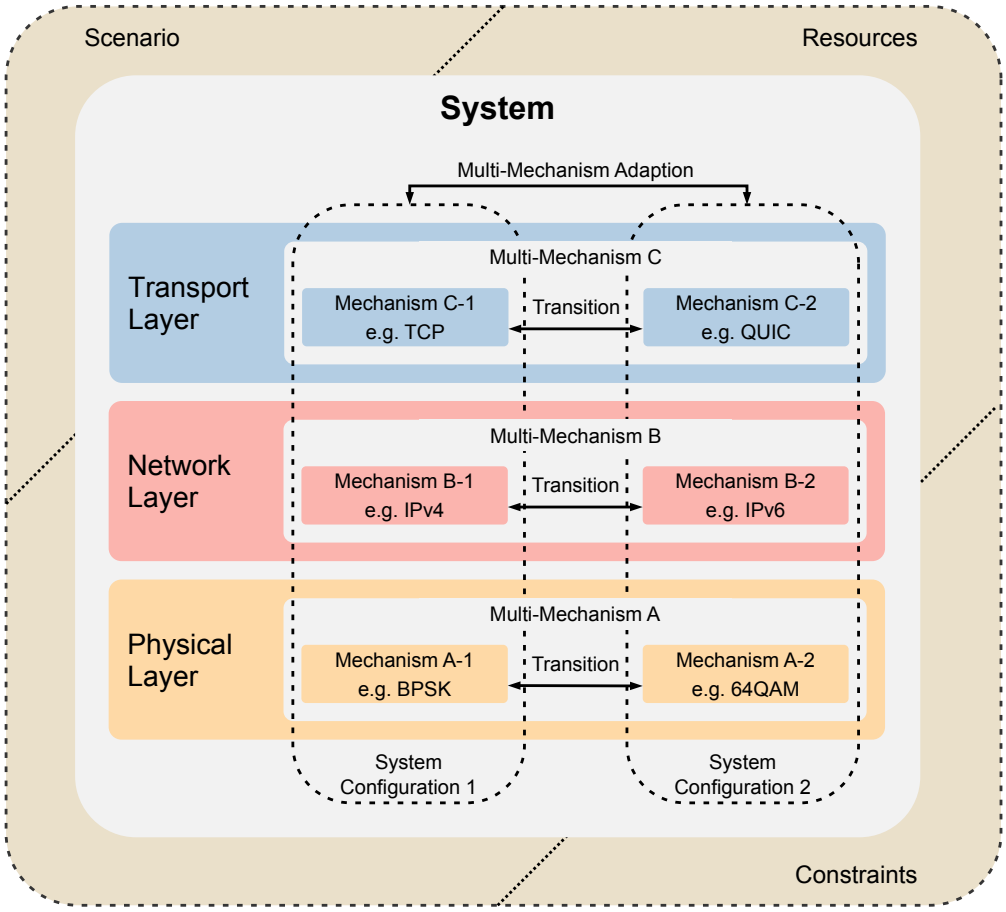


Figure 2.10: Exemplary multi-mechanisms in a communication system [105]

The same concept can also be applied to transitional computing, which is a computing paradigm that introduces new transition types that can be paired with existing transition types to extend the scope of mechanism transitions to a distributed computing system. Graubner [105] defines three different transitions with respect to computing: (i) transitions between locations, (ii) transitions between implementations and (iii) transitions between modes. Services and functionalities can be applied on different virtual or physical hosts in a distributed system. An example for transitions between locations is the relocation of a service between clouds in different geographical areas. Transitions between implementations provide functionally similar implementations of particular algorithms and their transitions. For example, face detection algorithms with different accuracies and runtime properties (e.g., less acurate but more efficient versus more accurate and less efficient) and transitions between them for changing scenarios can be considered as a transition between implementations. Finally, transitions between modes provide transitions between different software- and hardware layers within the same system.

2.7 Summary

This chapter introduced fundamental concepts, technologies, and terms. Different aspects of communication networks, two major network access technologies and fundamentals regarding the concept of the Internet of Things were discussed. Programming and data analysis concepts and the concept of transitions was presented. Each of these topics are essential for the upcoming chapters, since they are the building-blocks of many contributions throughout this thesis.

3

Situation-aware Edge Computing

In this chapter, the novel concept of situation-aware edge computing is presented. First, the terms situation and situation-awareness are introduced and defined in Section 3.1. The first subsection introduces situations as they are understood in the literature, followed by our own definition of situations. Two examples are presented to illustrate what situation means, followed by a paragraph that makes the differences between situation and context clear. The second subsection of Section 3.1 introduces the concept of situation-awareness and gives a definition that is used throughout this thesis. After that, a three-level approach to achieve situation-awareness is presented. This is followed by a clear distinction between situation-awareness and context-awareness.

Section 3.2 introduces edge computing, followed by defining the three relevant sub-areas infrastructure edge, device edge, and embedded edge. Finally, the remaining chapters of this thesis are categorized into the three presented sub-areas.

Section 3.3 formulates challenges for situation-aware edge computing, phrases open questions, and presents how situation-aware edge computing can be achieved and used for various applications. To achieve situation-aware edge computing, it is broken down into the concepts of situation-aware infrastructure edge computing, situation-aware device edge computing and situation-aware embedded edge computing. Within these areas, unique challenges and open questions are identified, whereas in the last subsection the approaches in this thesis are introduced together with a brief explanation about how these approaches contributed to the overall goal of situation-aware edge computing.

3.1 Situation and Situation-awareness

Situation-awareness originates from cognitive science, the interdisciplinary study of human mind processes. In the recent past, however, situation-awareness played an important role in the operation of complex non-human systems. Operators of such systems, but with respect to this thesis also the systems themselves, have to extract essential environmental information for their domain, rely on the derived knowledge of their environment, and project future process state changes. According to Endsley, system control can not be effective without this understanding and projection [76]. To achieve situation-awareness, the term situation must first be defined and clearly distinguished from the concept of context.

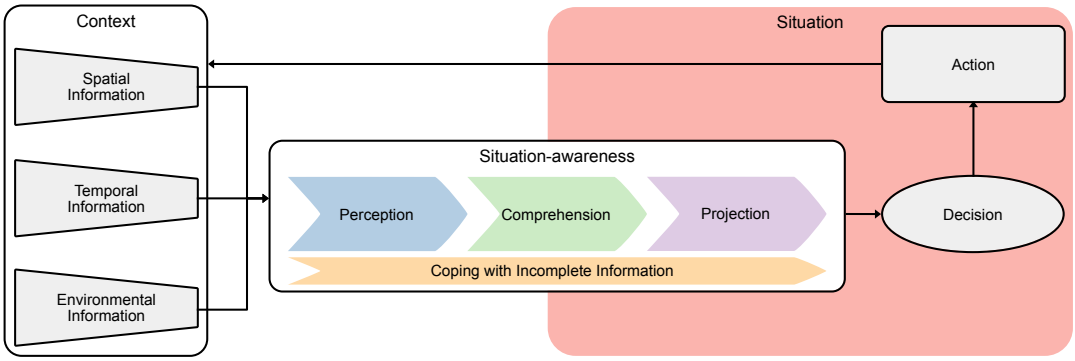


Figure 3.1: Overview of situation and situation-awareness

3.1.1 Situation

The term *situation* has many meanings in everyday use. On the one hand, it describes the environment in which one finds oneself, but also means spatial or temporal circumstances of an entity, i.e., persons or objects. In the literature, different definitions of situation can be found. For example, Abowed et al. define situation as the state that a system is in. This state contains all information that is required to characterize the situation [3]. Hossain et al. define a situation as a set of information and their conditions that exist at a particular time and that are important to the current application [115]. Meissen et al. view a situation as the semantic interpretation of perceived information [165], Ye et al. define a situation as a higher-level and domain relevant concept as a result of the interpretation of the perceived information [268]. Preden et al. use the term situation parameters for such information [196].

Definition

All of these definitions have in common that they are building on information that is perceivable. Further, a situation is domain or application specific and is an interpretation specific for the given entity, i.e., a situation may not be the same for different entities, although the surrounding information might be. Another important aspect for a situation is its time-dependency, i.e., a situation is only valid for a given time, although the situation might be the same in the upcoming time periods.

The term situation is defined as follows in the scope of this thesis:

Definition 3.1.1: Situation

A *situation* is the state of an entity that is based on perceivable information, valid for a given time in a specific domain or application, and interpreted for the given domain.

Figure 3.1 visualizes the term situation in relation to information, denoted as context. A situation is derived from the perceived spatial, temporal, and environmental information, but also from other information sources.

Example

Consider the example of an emergency, such as a natural disaster. During a disaster, a large body of information is available. Early warning systems might have sounded the alarm, various images are available like satellite imagery or pictures provided by affected people, authorities provide evaluations and forecasts of upcoming events such as floods or buildings in danger of collapse, or up-to-the-minute reports from rescue helpers on site. Different rescue helpers like firefighters, medical helpers or security forces have all the same information regarding the disaster. All this information is perceived by mission control personnel from the different areas (i.e., the respective heads of the firefighters, medical helpers and security forces) and interpreted. The interpretation, i.e., the derived state from the information is then called situation. Since a situation is domain-specific, or in this example specific for each area of rescue helpers, firefighters, medical helpers and security forces derive a different situation. The situation of firefighters describes the number, size, and spread of fires, while the situation of medical helpers revolves around the number and extent of injured people in the crisis area. All rescue helpers might have the same information available regarding the natural disaster that occurred and its effects, but the derived situation is specific to their own domain. Another example in the edge computing domain is a user's situation of leaving their home. In this situation the user's smartphone is about to lose the Wi-Fi connection so that a vertical handover to a cellular connection is required. The available information here is for example the received signal strength of the Wi-Fi connection, the sensor readings from the device's accelerometer and the device's state like whether headphones are connected. This information is comprehended and the user's situation is interpreted during the comprehension, i.e., when the user is about to leave home.

Distinction of Situation and Context

In the introduction of this Subsection 3.1.1, it was mentioned that the term situation has different meanings in everyday usage. One of these meanings may be the synonymous use of the terms situation and context. However, it is important to note that in the literature, but of course also for this thesis, these two terms are not interchangeable, but indicate different concepts. Context denotes all the information surrounding a situation, as shown in Figure 3.1. Thus, it is possible that the same context might result in different situations for different entities. Firefighters and medical helpers might have the same information regarding the natural disaster that occurred, but their respective situation is different. Therefore, a situation is the continuation of context. If the context is perceived and a higher-level, domain-relevant concept is derived, this derivation is called situation. Further, the context, i.e., the surrounding information, is not a singular state but a continuous stream of information, whereas the situation is the state that follows the interpretation of the entity. To conclude, a situation is a distillation of the context, i.e., the result of the analysis of contextual information important for the domain and the application at a given time.

3.1.2 Situation-awareness

The term *situation-awareness* originates from cognitive science, the interdisciplinary study of the mind of humans, i.e., cognition, intelligence, and behavior. It involves psychology,

neuroscience, anthropology, and philosophy to how the human mind works. [242]. Thagard defines situation-awareness as "[...] understanding, of a dynamic environment" [242] and further "as a system of cognitive processes" (Durso et al. [71]). To achieve situation-awareness, it is required to perceive information and to integrate derived semantic information to form a representation of a situation [71].

In the recent past, the concept of situation-awareness was adopted in various fields of computer science. For example, in the security domain it is used to build secure IoT applications [278], detect network attacks [57, 117, 187] or authenticate devices in a local network [261]. Further, situation-awareness is used in emergency response applications [133], autonomous driving [171] and aviation [181] or UAV environments [14, 157].

Furthermore, military researchers applied situation-awareness in their field. Endsley [75] encountered that pilots of combat aircraft used their own mental situation-awareness model. She proposed important considerations to design aircraft that support the pilot's situation-awareness. Endsley proposed a three-level approach, perception, comprehension, and projection that is widely accepted among researchers in various fields [75]. In the remainder of this thesis, these three levels are used to achieve situation-aware edge computing, with an additional step, coping with incomplete information, added later in this section and visualized in Figure 3.1.

Perception First, environmental information has to be perceived to achieve situation-awareness. Analogous to Definition 3.1.1, perception means recognizing information in the current environment. This can be any information to characterize a situation, i.e., the context in which the situation occurs.

Comprehension The second level in achieving situation-awareness is comprehension. Here, the perceived information is to be synthesized into higher-level concepts that define the current situation. The synthesis is achieved by analyzing and evaluating the perceived information collected in the first level, perception. The goal of comprehension is to give a comprehensive understanding of the situation. At this point, a situation-aware system does not deal with context information anymore but with situations.

Projection The last of the three levels is projection, i.e., predicting the situation in the near future. This step is essential in the decision-making process, which is the ultimate goal in achieving situation-awareness. It means that based on the current situation, the situation in the near future can be projected so that it can be decided what actions or steps have to be executed based on the goal of the system or application. This can also be seen in Figure 3.1, where the projection is leading to a decision that in turn triggers an action.

Incomplete Information One major missing point in the literature is the question of how to deal with incomplete information. Here, incomplete information can not only mean that information is missing, but also that information is inaccurate or uncertain, e.g., from untrusted parties, faulty sensors or inaccurate measurements. Moreover, the problem of incomplete information does not only concern, as one might misunderstand, the aspect of perception. Incomplete information can lead to a wrong or only incomplete comprehension of

the situation being obtained, which in turn can lead to wrong projections and thus to wrong or non-optimal decisions. Therefore, it is essential to incorporate incomplete information in the above presented three-level approach to achieve situation-awareness, which is proposed as a fourth level in this thesis (cf., Figure 3.1).

Definition

Based on the definitions from the literature and the above presented four levels to achieve situation-awareness, situation-awareness in this thesis is defined as follows:

Definition 3.1.2: Situation-awareness

Situation-awareness is the process of perceiving information in time and space, comprehending the information to derive a situation, and projecting future situations.

In simple words, situation-awareness is about understanding what is going on, so the system can decide what to do. It is important to understand that a triggered action results in changing the context so that it is crucial to apply these three steps continuously.

Distinction of Situation-awareness and Context-awareness

Similar to the distinction of situation and context, the difference of situation-awareness and context-awareness is of equal importance. As described in Subsection 3.1.1, context refers to all information surrounding an entity, whereas situation is the state of the entity that is derived from the context. The same distinction has to be made for situation-awareness and context awareness. Context-awareness can be defined as the awareness of every piece of information in a specific environment. Situation-awareness, on the other hand, is to be aware of information relevant for the domain and its interpretation. Context-awareness is the knowledge about the environment, situation-awareness is the understanding of the environment. Considering the natural disaster example from above, context-awareness would be the affected persons' awareness of all information available in the emergency, although it might not be relevant for their domain, such as information regarding a flood in another part of the city. Situation-awareness is the awareness of the derived state from domain-relevant information, such as size and spread of fires in the affected disaster area.

3.2 Edge Computing

Section 2.5 introduced the concept of edge computing as a paradigm that enables data processing and storage to be decentralized and provides an alternative to cloud computing. A wide variety of hardware, execution environments, and degrees of decentralization are united under a single concept. This heterogeneity and its challenges require the field of edge computing to be considered in a more fine-grained way to be able to develop optimal solutions and approaches in the area of situation-aware edge computing. Therefore, this section defines the terms *infrastructure edge, device edge, and embedded edge*. Figure 3.2 visualizes the hierarchical organization, the location in the stack of edge computing of the three different

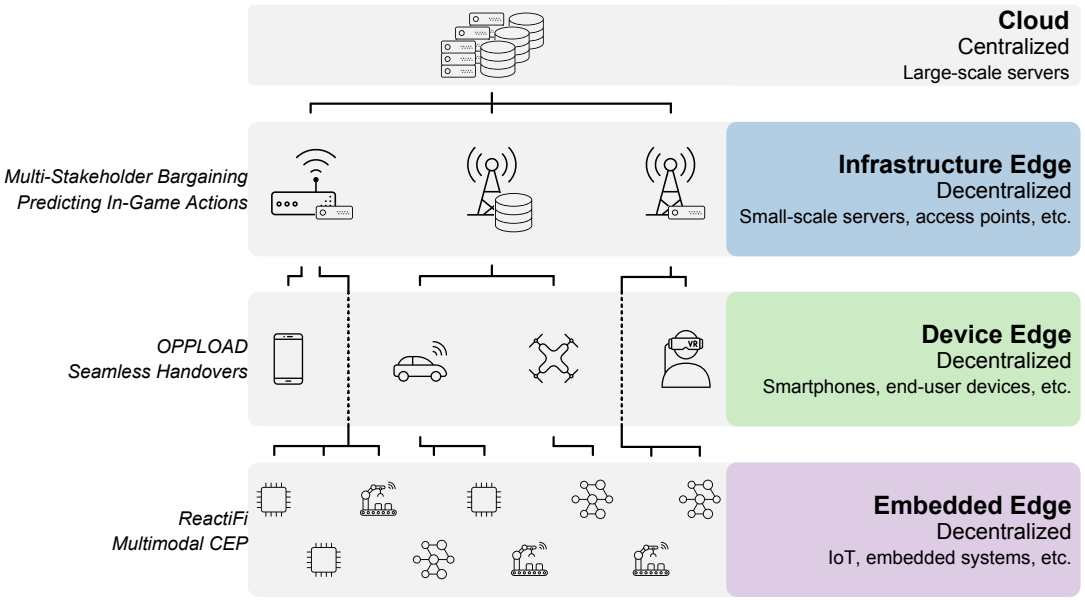


Figure 3.2: Hierarchical structure of edge computing locations

areas and how they are connected to each other. To the left of Figure 3.2, the chapters of the remainder of this thesis are grouped into the respective levels to which they make a significant contribution.

3.2.1 Infrastructure Edge

The *infrastructure edge* (blue layer in Figure 3.2) is mainly driven by telecommunication providers. By providing storage and computation resources at the edge of the network, i.e., at base stations or small-scale servers called *cloudlets*, application providers can place processes on compute nodes that are usually located in a one-hop distance to their users instead of the cloud. These facilities achieve low latencies due to the proximity to users. Furthermore, they are equipped with decent computing capabilities making it possible to execute rather complex computations.

Definition 3.2.1: Infrastructure Edge

Infrastructure edge refers to small-scale compute facilities located on or near infrastructure at the edge of the network provided by infrastructure providers. Examples are base stations or small-scale servers. They have sufficient power supply and rather decent computing and storage capabilities, but they significantly more limited than a central cloud infrastructure. For the remainder of this thesis, compute facilities at the edge of the network will be called infrastructure devices.

3.2.2 Device Edge

The *device edge* (represented as the green layer in Figure 3.2) incorporates mobile devices like mobile phones or notebooks as compute resources. This makes it possible to execute processeses in local networks without relying on infrastructures of telecommunication or cloud providers. Although this facilitates novel applications and supports decentralized computing approaches, devices used for device edges are often faced with hurdles not found in infrastructure edge environments. These mobile devices are often constrained in terms of computational power as they are usually mobile phones or other small computers. Further, in many cases, the devices used for device edge computing are battery powered and have a limited power supply. Thus, applications using capabilities of such devices have to take these restrictions into account and handle them accordingly. In contrast to infrastructure edge, providers of services are usually not involved on the area of device edge, but devices either offload processes to other devices or execute them locally. Processes are rarely offloaded to infrastructure devices or the central cloud.

Definition 3.2.2: Device Edge

Device edge refers to mobile devices like smartphones as computing devices (mobile devices from now on) used in local networks for processes execution. These devices are often constrained in computational power, storage, and power supply.

3.2.3 Embedded Edge

With the advent of the Internet of Things (IoT) paradigm and corresponding devices like sensors and embedded systems, the amount of data produced within local networks increased significantly. This introduced additional burdens on the network's backend and central cloud infrastructure. However, much of the produced data like readings from sensors are not necessarily required by the end user applications as they are more interested in information derived from the data. For example, a user might not be interested in the raw values of a gas sensor but only in the air quality index derived from the gas values. The *embedded edge* paradigm (purple layer in Figure 3.2) is a way to facilitate computations on IoT- and embedded devices so that only preprocessed and aggregated information are consumed by the user or sent to upper layers of the computation stack. Although embedded systems may be equipped with sufficient power supply, they are in general very limited in terms of computational power and available storage. Thus, the embedded edge is quite resource-constraint and only able to execute specific applications.

Definition 3.2.3: Embedded Edge

Embedded edge refers to IoT devices and embedded systems as computing devices (embedded devices from now on) usually used to preprocess and aggregate sensor readings directly on the embedded device itself. Embedded devices often have sufficient power supply but only very limited computation power and storage abilities.

Table 3.1: Overview of different computing paradigms

Feature	Cloud	Infrastructure Edge	Device Edge	Embedded Edge
Operator	IT companies	Telecommunication providers	IT companies, private entities, individuals	
Hardware	Large-scale servers	Base stations, small-scale servers	End user devices, mobile devices	End user devices, IoT devices, embedded systems
Architecture	Central	Decentralized, distributed, hierarchically organized		
Mobility	No	Yes		
Latency	●○○	●●○	●●●	
Power Supply	●●●		●◐○	●○○
Computation	●●●	●●○	●◐○	◐○○
Storage	●●●	●◐○	●○○	○○○

3.2.4 Overview, Comparison, and Integration

The term *edge computing* summarizes the approaches of infrastructure edge, device edge and embedded edge. It is used to distinguish centralized cloud approaches from decentralized computation facilities, even though infrastructure edge, device edge and embedded edge describe different computing layer in the domain of edge computing. Table 3.1 gives an overview of the different layers. In summary, infrastructure edge describes compute facilities that are operated by telecommunication providers at the edge of the network on devices like base stations or small-scale servers. They offer virtually unlimited power supply, whereas computation and storage are more limited compared to cloud computing. Device edge refers to decentralized approaches operated by IT companies, private entities or individuals beyond the edge of the network like smartphones. They are mobile, have low latency, whereas power supply, computation and storage capabilities are limited. Finally, embedded edge describes approaches where devices like IoT- or embedded systems are organized in a decentralized manner and operated by the same categories of providers like device edge. They support mobility and achieve low latency. Power supply and computation capabilities are very limited, and storage is often not available at all. Finally, in contrast to edge computing, cloud computing facilities are operated by IT companies in the network core and are usually centrally organized large-scale servers with no mobility support. They offer virtually unlimited power, computation and storage, at the cost of high latencies.

The works this thesis is based on are categorized according to the above definitions:

Infrastructure Edge The proposed *Multi-Stakeholder Bargaining* approach in Section 4.2 copes with incomplete information for placing services on cloudlets, i.e., compute hardware operated by telecommunication providers, to reduce operating costs. Thus, it is located in the area of infrastructure edge. *Predicting In-Game Actions* (cf., Section 4.3) addresses the area of infrastructure edge, as it also aims to comprehend when and why users play mobile AR games to find suitable cloudlet locations.

Device Edge *OPPLOAD*, presented in Section 5.2, operate on smartphones and is used in opportunistic networks, which is centered around the device edge paradigm. *Seamless Handovers* to predict when users are going to lose their Wi-Fi connection, as discussed in Section 5.3, to proactively hand over to a cellular connection like LTE are performed on smartphones. Thus, they are located in the device edge.

Embedded Edge *ReactiFi* (cf., Section 6.2) is a novel programming language based on the reactive programming paradigm. It enables embedded devices within off-the-shelf end user devices to perceive PHY and MAC layer information and to comprehend the situation. *Multimodal CEP*, presented in Section 6.3 executes CEP queries on different modes, i.e., execution environments such as coprocessors and embedded systems in off-the-shelf end user devices. These two approaches are in the embedded edge area.

3.3 Design

For situation-aware edge computing, several challenges have to be overcome, and open questions have to be answered. All three areas of edge computing, i.e., infrastructure edge, device edge and embedded edge have their own set of hurdles. For example, in the area of infrastructure edge economic considerations have to be taken into account. For the device edge, it has to be discussed how to cope with the heterogeneity mobile devices. The area of embedded edge needs an appropriate programming paradigm for achieving situation-awareness. Therefore, the remainder of this section separately identifies the challenges of all three areas with respect to situation-awareness and formulates open research questions and answers that are essential for addressing these challenges.

3.3.1 Situation-aware Infrastructure Edge Computing

Challenges The challenges in infrastructure edge computing is categorized into two areas. First, technical considerations and second economic and corporate policy aspects.

The technical issues are widely solved and well researched, since the ideas of edge computing can be dated back to the 1990s. Further, the ETSI started a standardization process to define the technical aspects of edge computing, culminating in multi-access edge computing, as described in Subsection 2.5.4. For example, the computing platforms (i.e., infrastructure devices) are centered around a so-called application servers, that are either realized as a virtual machine or containers integrated into cellular networks and offer computing resources, storage capacity and access to the radio network. The cellular network should then be opened, e.g., using APIs, for third-party service providers so that they can deploy their services. Finally, the servers should be either located directly at the base station (e.g., the e-NodeB in LTE networks) or at other locations that are in proximity of the base station.

The second challenging area of infrastructure edge computing, economic and business aspects, are almost unexplored. The first challenge in this area is how to decide where to place the service, i.e., how to decide which of the countless infrastructure devices should be used. Or maybe the service providers should use all available infrastructure devices at the same

time? Although this challenge is tackled in the literature from the technical perspective, it is still largely unclear how to deal with this issue from an economic perspective. The Internet consists of multiple stakeholders, i.e., parties (mostly companies) that operate the Internet. These stakeholders, mainly infrastructure providers (IPs, i.e, companies operating the infrastructure devices) and service providers (SPs, i.e., companies offering services to users), are usually companies that have economic requirements, constraints and goals. With situation-awareness, SPs can find infrastructure devices that fulfill their requirements and have the potential to be financially lucrative. Another open challenge is about how IPs and SPs should find an agreement with respect to the price the SP should pay for using an infrastructure device. As noted, both IP and SP are companies that are unable or not willing to reveal all required information to the respective other stakeholder that is required to agree on a price, because, for example, the corresponding valuation of the other stakeholder is not known. Here, again, situation-awareness plays a crucial role because based on a plethora of contextual information like public price lists of competitors or environmental information (e.g., weather), the stakeholders can project their future situation with respect to where users will likely use the service and make economic decisions based on their situation.

Open Research Questions

- How can IPs and SPs achieve situation-awareness without relying on their mutual proprietary information?
- Which environmental information can be perceived and used to comprehend a situation with respect as to where users are using a service to find suitable infrastructure devices to place services on?
- How can both providers comprehend their situations without knowing their mutual valuation of the infrastructure device?
- Based on the perceived information, how can the service provider comprehend its situation with respect to where and how users use its service?
- How can IPs and SPs find an agreement regarding the price of placing a service on infrastructure devices with incomplete information?
- Can both providers reliably project their future situation with respect to their cost reduction albeit both have to cope with incomplete information?
- How accurate can the service provider project the service usage for particular infrastructure devices?
- How can the IP and SP make confident decisions on which infrastructure devices to place a service on, if they both have to cope with incomplete information?
- Are the placement decisions accurate with respect to the projected service usage situation from the service providers' view?

Chapter 4 will address the challenges and open questions separated into two approaches, *Multi-Stakeholder Bargaining* dealing with incomplete information and its effects from the IP's and SP's view and *Predicting In-Game Actions* for the questions regarding the service provider view dealing with users.

3.3.2 Situation-aware Device Edge Computing

Challenges In the area of device edge computing, two main challenges have to be addressed. The first one is about how off-the-shelf end user devices like smartphones can be used as devices for executing processes. The second challenge is dealing with device mobility, more specifically intermittent connections and unpredictable disconnections both between mobile devices and cellular networks but also between the mobile devices themselves.

Regarding the first challenging area, it is, for example, unclear how to decide which surrounding mobile devices are suitable for computations. In scenarios where the device edge computing paradigm is beneficial (e.g., for emergency response), it would be counter-productive to use mobile devices that are not able to compute a given process. This can lead to either overloading the chosen mobile device or to at least using a non-optimal mobile device. Thus, the goals of device edge computing, i.e., executing processes faster or more efficiently than executing it locally, would not be achieved. Further, another crucial part is resource preservation. As was identified in Section 3.2.2, devices in device edge are usually mobile devices like smartphones, laptops but also drones, for example. What they all have in common is that they are usually battery powered, i.e., their amount of energy is limited. Thus, it is important to preserve as much energy as possible. Here, situation-awareness helps to identify suitable mobile devices. The problem, however, is that it is not possible to reliably determine by perceiving environmental information whether a mobile device has sufficient computation or memory resources, or how much energy is available so that a strategy is required to cope with this incomplete information. Finally, mobile devices are highly heterogeneous. For example, they are using different operating systems (iOS, Android, Linux, etc.), different CPU architectures (x86, ARM, 32 Bit, 64 Bit, etc.) and have different hardware components (GPUs, coprocessors, DPSs, etc.). Thus, a generic and situation-aware execution environment is necessary to empower nodes to serve as devices for device edge computing.

The second question imposes challenges regarding the connectivity of mobile devices and how to cope with the mobile devices' mobility and intermittent connections. If a process is offloaded to another mobile device and the connection between the devices is disrupted, it might not be possible to return the result of the process back to the offloading mobile device. Thus, it is crucial to detect possible connection losses in advance in order to be able to take precautions in time, such as setting up a second connection in parallel that does not drop out because it uses a different wireless technology. To do so, a situation-aware approach can contribute essentially by comprehending the local mobile device's situation and project the future connection situation, i.e., whether the connection is lost or not. In addition, another way to deal with connection interruptions is to use network protocols and technologies that go beyond the regular TCP/IP stack. Opportunistic- and disruption-tolerant networks should be mentioned here in particular, since these two technologies are especially suitable for challenging networks due to their network topology on the one hand and the use of unconventional application layer protocols on the other. With situation-awareness, it is possible to use mobile devices for offloading processes onto where the likelihood of getting a result is maximized despite a high mobility of devices.

Open Research Questions

- How can a mobile device be identified as suitable, if the perception of environmental information is not sufficient?
- How can mobile devices announce their capabilities in an efficient yet effective manner so that they can be identified as suitable?
- Which perceived information of a device can be used to comprehend the connection situation?
- What identifies a device as suitable so that the projection of the future situation can reject unsuitable mobile devices?
- How can the situation of a device with respect to the connection situation be comprehended accurately?
- How can the projection identify the most suitable mobile device to offload a process to?
- Which model can confidently project the future connection loss based on the comprehended connection situation?
- How can the situation with respect to resource usage and energy depletion of mobile devices be projected?
- What is the decision basis for offloading a process if resource utilization can not be projected correctly?
- Based on the projected connection situation, which action is suitable to cope with the situation of connection loss?

Challenges and open questions regarding situation-aware device edge computing will be addressed in Chapter 5. *OPPLOAD* deals with the questions with respect to identifying suitable mobile devices, whereas *Seamless Handovers* uses situation-awareness to project if a mobile device will lose its connection.

3.3.3 Situation-aware Embedded Edge Computing

Challenges In the field of embedded edge computing, the challenges are not in the areas of economic or corporate politics, since here several independent companies do not have to negotiate for a price and usage, as is the case with infrastructure edge computing. Furthermore, it is not necessary to run arbitrary processes on heterogeneous devices that also only have a finite energy capacity and are highly mobile, as is the case with device edge computing. The challenges in the area of embedded edge computing are primarily that common programming and data analysis approaches are built on programming paradigms that are not appropriate for the goal they try to achieve. The main application areas of embedded edge computing are either sensor networks, where data is aggregated and preprocessed, or general-purpose computations of small processes like in the area of IoT. What both application areas have in common, however, is that they are event- and data-driven. But the predominant programming paradigms for embedded edge computing are imperative and procedural (especially the C programming language), which requires a lot of callbacks, data references, and opaque call graphs to process the incoming events and data streams. The use of inappropriate languages makes these systems error-prone and unreliable, which gets in the way of easy-to-implement, situation-aware, embedded edge computing. It is also important that not every area of embedded edge computing should apply the same programming paradigm. Depending on the use case, two paradigms are well-suited for particular tasks.

The first one is complex event processing (CEP). CEP is suitable, for example, for scenarios in which a data stream has to be aggregated or preprocessed like it is the case in sensor networks or similar applications. The second paradigm is reactive programming, a paradigm that enables developers to implement general-purpose computations that are centered around processing data streams. Reactive programming is especially useful whenever arbitrary general-purpose computations should be implemented that rely on analyzing and processing of continuous data streams. These paradigms and the concrete implementations should be able to support situation-aware embedded edge computing in the first place. For example, it should be easily possible to perceive various, application specific, environmental information and add functionality that supports the comprehension of the situation like filtering data or aggregating multiple data streams. Further, the projection should also be supported so that decisions can be made about which action to execute. Additionally, it should be possible to actually execute actions based on the projection.

Open Research Questions

- How can interfaces be defined that support the perception of information that arrives as a stream of data for CEP?
- How can reactive programming be used to perceive environmental information?
- Which aggregations are required to allow situation comprehension using CEP?
- Which language primitives have to be supported for comprehending situations in the area of reactive programming?
- How can CEP support the projection of future situations to achieve situation-awareness on embedded devices?
- Which projection functionality can be supported on embedded devices to aid developers in making decisions based on the reactive programming paradigm?

In Chapter 6, the open questions and challenges of situation-aware embedded edge computing will be addressed. *ReactiFi* introduces a programming language following the reactive programming paradigm enables developers to implement situation-aware applications on embedded devices. *Multimodal CEP* presents a CEP approach supporting situation-aware embedded edge computing in sensor networks or areas using embedded devices.

3.3.4 Situation-aware Edge Computing

To fully achieve situation-aware edge computing, the above presented challenges have to be addressed and the open research questions to be answered. Therefore, Table 3.2 lists the different approaches examined in this thesis, structured according to the three areas of *Situation-aware Infrastructure Edge Computing*, *Situation-aware Device Edge Computing*, and *Situation-aware Embedded Edge Computing*.

3.4 Summary

This chapter introduced the novel concept of situation-aware edge computing. The terms situation and situation-aware were defined, followed by the definitions and classifications of

Table 3.2: Contributions to situation-aware edge computing

Situation-aware Infrastructure Edge Computing	
Multi-Stakeholder Bargaining	An approach where situation-awareness supports the economic decision about which infrastructure devices a service should be placed on despite incomplete information.
Predicting In-Game Actions	Using situation-awareness to decide where to place services based on where users use services.
Situation-aware Device Edge Computing	
OPPLOAD	Suitable mobile devices to execute a process are identified using situation-awareness by perceiving information announced by mobile devices, comprehending their situation and projecting the future situation with respect to the best suitable device for the given process.
Seamless Handovers	This approach uses the situation of a mobile device to project whether the connection will be lost or not. Depending on the projection, a second connection using an alternative cellular network is established.
Situation-aware Embedded Edge Computing	
ReactiFi	An approach that proposes a reactive programming language that enables developers to implement situation-aware applications on embedded devices.
Multimodal CEP	Introducing a CEP approach that supports situation-aware and event based applications for embedded devices.

the concepts of infrastructure edge, device edge and embedded edge.

Afterwards, the concept of situation-aware edge computing was broken into three parts:

- situation-aware infrastructure edge computing,
- situation-aware device edge computing, and
- situation-aware embedded edge computing.

While the technical aspects of situation-aware infrastructure edge computing are largely researched, the economic and business policy aspects are not. In the area of situation-aware device edge computing, it has to be investigated how to identify suitable mobile devices using the concept of situation-awareness and how to project their future situation with respect to the connection state, i.e., if the connection of a mobile device is about to break. Finally, for embedded edge computing, new programming paradigms have to be applied to support developers to implement situation-aware applications on embedded devices. The proposed approaches to address these challenges are presented in Table 3.2.

4

Situation-aware Infrastructure Edge Computing

This chapter presents two approaches using situation-awareness in the area of infrastructure edge computing. Section 4.1 motivates the use of situation-aware infrastructure edge computing, while Section 4.2 shows an approach where situation-awareness aids to support economic decisions about where to place services on infrastructure devices. The focus of this approach is on dealing with incomplete information during placement decisions. Section 4.3 presents a method that uses situation-awareness to comprehend the service provider's situation with respect to where users use services to decide on which infrastructure devices a service should be placed. Section 4.4 concludes this chapter.

Parts of this chapter have been published previously [81, 236].

4.1 Motivation

In infrastructure edge computing environments, service providers deploy their services on infrastructure devices, such as small-scale servers operated by telecommunication providers, at the edge of the network. Such infrastructure devices operate in the proximity of users to improve service quality, to reduce latency, and to increase throughput [159, 224], but also to reduce the service providers' operational cost [253], since the amount of communication from the edge through the network core can be reduced, e.g., in online gaming contexts up to 95% [253]. Decision-making regarding which infrastructure device should be used to place a service on is significantly improved. The novel approach of situation-aware infrastructure edge computing enables stakeholders of the Internet, i.e., companies and institutions offering services, providing compute and storage resources, and operating networks, to make economic decisions using their situations, which are built on information of their users, of other stakeholders, and also of themselves. The decision about where a service should be placed depends on information of infrastructure- and service providers, but also on information of the users' of the service. Furthermore, as the stakeholders of the Internet are independent entities, they have to cope with incomplete information, e.g., because companies are not willing or able to share their private information with other companies.

Figure 4.1 depicts an infrastructure edge computing environment, where a service provider wants to place its service in an economic reasonable manner. Thus, the service provider has to be aware of its situation with respect to (i) how much the usage of infrastructure devices will cost and (ii) when and where its users will use the service. For (i), the service provider

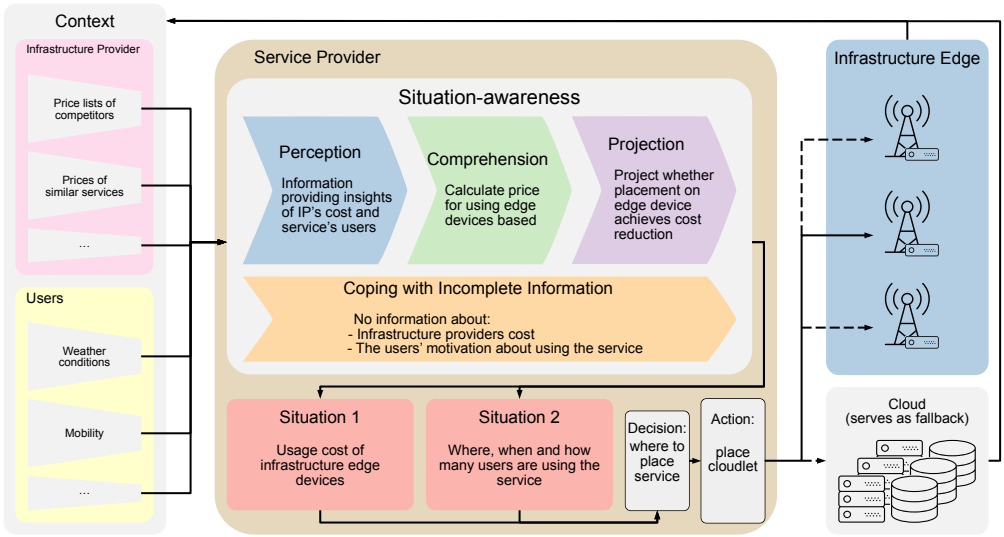


Figure 4.1: Situation-aware infrastructure edge computing

perceives information of the provider of the infrastructure device, called infrastructure provider, with respect to the valuation of the infrastructure device. Since the infrastructure provider is not able or willing to tell its valuation, the service provider relies on secondary information like public price lists of competitors or companies offering similar compute infrastructures. Based on the perceived information, the service provider comprehends the infrastructure provider's valuation and can project the cost incurred by using the infrastructure device. However, to fully project the cost for using an infrastructure device, the service provider has to know how many users will use its service. Therefore, the service provider additionally has to perceive information of the end users and comprehend their situation, to project when and where users use the service, which corresponds to part (ii). Based on both situations, the service provider decides where to place a service and triggers the corresponding action. The service can either be placed in the cloud, e.g., in the case where the infrastructure devices are too expensive, or on the infrastructure devices otherwise. If an infrastructure device should be used, the service provider also decides on which of the available devices the service should be placed. In Figure 4.1, the dashed arrows indicate possible targets of the placement, although the service provider chose the infrastructure device with the solid arrow. Finally, the placement influences information so that the entire procedure will be evaluated periodically.

4.2 Multi-Stakeholder Service Placement via Iterative Bargaining with Incomplete Information

To achieve the goals of improved service quality and reduced cost, the use of *cloudlets* has been proposed [210], which are small-scale datacenters located at the edge of the Internet in one-hop proximity to users. In this section, the following stakeholders of the Internet are considered: (i) infrastructure provider (IP), (ii) service provider (SP), (iii) cloud provider, and (iv) users. The IP (e.g., AT&T, Vodafone, etc.) owns and operates wireless access networks consisting of base stations (BSs), the network backend, as well as the infrastructure devices such as cloudlets. The SP (e.g., Netflix, Niantic, etc.) offers a service, such as a mobile AR game or a high-quality video stream, to users. Reducing network traffic is beneficial for both IP and SP, since this reduction results in less expenses for using cloud services for the SP, and lower operational cost for the IP's network. Hence, for cost-optimal service placement, the IP and the SP are equally important, and both stakeholders should be involved to decide on which cloudlet a service should be deployed. A novel bargaining approach in which the IP and SP bargain for cost-effective service placements incorporating incomplete information about the respective other stakeholder in the bargaining is presented. Being aware of their situation, e.g., by using publicly available information like public price lists or scientific papers about pricing cloudlets, both stakeholders can project their future situation with respect to their own cost and the cost of the respective other stakeholder and use this projection to decide where to place cloudlets.

The remainder of this section is organized as follows. In Subsection 4.2.1, the service placement scenario and system model is introduced. Subsection 4.2.2 analyzes the bargaining process with complete information, and Subsection 4.2.3 presents the solution to the bargaining problem with incomplete information including two variants to discuss different scenarios of incomplete information. Experimental results based on the collected real-world data set are provided in Subsection 4.2.4. Finally, Subsection 4.2.5 reviews related work.

Parts of this section have been published previously [236].

4.2.1 System Model

Figure 4.2 provides an overview of our system model. In this scenario, four stakeholders interact with each other: service provider (SP), infrastructure provider (IP), cloud provider, and users. However, we only consider IP and SP as active parts, since only these two stakeholders have to jointly find suitable cloudlets for placing services on them. Users initiate service requests whenever they use a functionality of a service. Users want a high Quality-of-Experience (QoE) of a service, which is achieved by handling service requests on nearby cloudlets. A cloud serves as the default service placement location, where all service requests can be handled. Thus, users and the cloud provider are not active parts in our system model.

The IP operates base stations (BS) and a network backend, offers network access via its BSs, and deploys and operates permanently available cloudlets. The cloud is accessible via the IP's network backend. Furthermore, BSs provide ubiquitous radio coverage and network access for users, and multiple users can be connected to one BS. By default, the service is deployed in the cloud with sufficient capacity to handle the requests of all of its users.

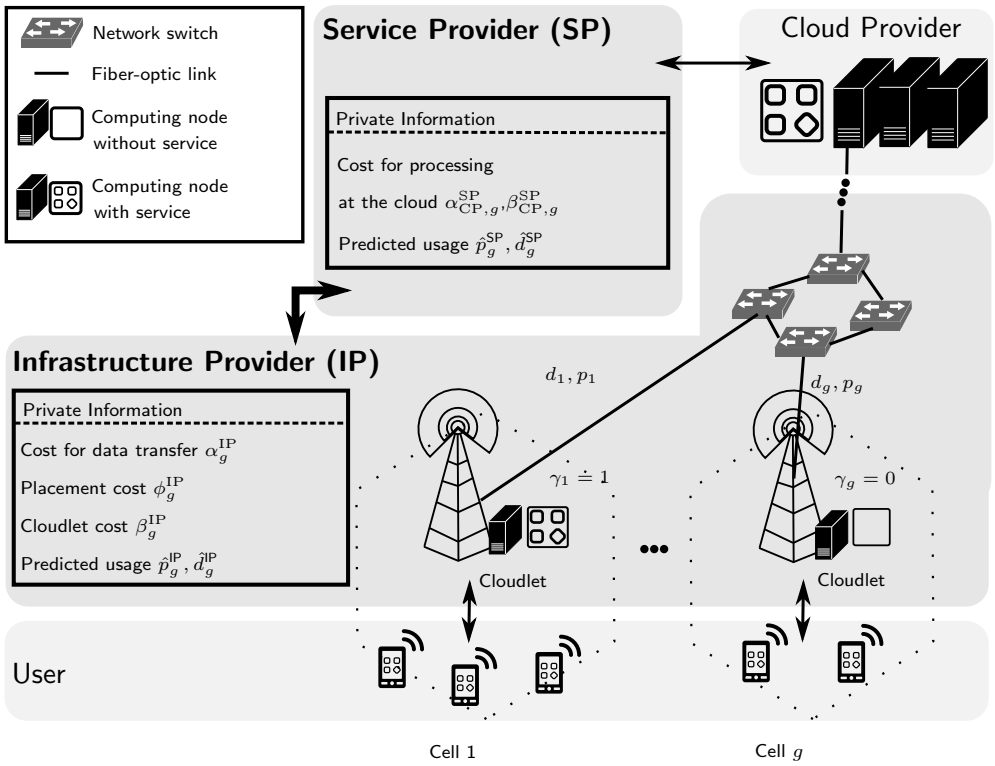


Figure 4.2: Overview of the system model

The SP may rent resources on the IP's cloudlets and place the service on them by paying a monetary compensation. The service is placed on the cloudlet by offloading relevant parts of the service from the cloud to the cloudlet. If the service is not placed, e.g., because the SP and IP cannot agree on a cost, it remains in the cloud.

Furthermore, we assume that the network can be divided into a grid G of cells with one cloudlet per cell $g \in G$. For each cell $g \in G$, we introduce an indicator variable γ_g, which is 1 if the service is placed on the cloudlet and 0 otherwise. The goal is to place the service on cloudlets that are advantageous for both IP and SP in terms of cost reductions.

Service Provider

The SP provides a service to users which requires computation and data transfer between the users and the cloud. There are two different options how the service requests can be handled, one of which is fixed for each cell g. First, the service requests are handled by the cloud. After the service requests arrive at the BS, the data of size $\tilde{d}_g$ data units is transferred from the BS to the cloud. The computation is done in the cloud, which requires p_g processing units.

The second option is that the service requests are handled by a cloudlet in cell g. After the service requests arrive at the BS in cell g, they are handled by the cloudlet located in cell g, which requires p_g processing units on the cloudlet. In addition to the cloudlet, the SP needs to use cloud resources and the IP's backbone network, e.g., in mobile AR games for

Table 4.1: Mathematical notations

Notation	Description	Notation	Description
$g \in G$	Cell g in the grid G	α_g^{IP}	Cost factor for every data unit incurred for the IP in cell g
$\gamma_g \in \{0,1\}$	Indicator variable for placement in cell g. $\gamma_g = 1$ if a service is placed in g, zero otherwise	$\alpha_{\mathrm{CP},g}^{\mathrm{SP}}$	Cost factor for every data unit the SP has to pay to the cloud in cell g
$\kappa^{\{\mathrm{SP,IP}\}}$	Total cost for SP or IP	$\beta_{\mathrm{CP},g}^{\mathrm{SP}}$	Cost factor for every processing unit the SP has to pay to the cloud in cell g
$\kappa_g^{\{\mathrm{SP,IP}\}}$	Cost incurred for SP or IP if the cloudlet is used in cell g	β_g^{IP}	Cost factor for every processing unit incurred for the IP in cell g
$\kappa_{\mathrm{IP},g}^{\mathrm{SP}}$	Cost that the SP has to pay to the IP for using the cloudlet in cell g	d_g	Data units from the cloudlet to the cloud (and vice versa) if service is placed in cell g
$\tilde{\kappa}_g^{\{\mathrm{SP,IP}\}}$	Cost incurred for SP or IP if no cloudlet is used in cell g	$\tilde{d}_g$	Data units from the BS to the cloud (or vice versa) if no service is placed in g
ϕ_g^{IP}	Fixed cost for placing a service on a cloudlet in cell g incurred for the IP	p_g	Processing units (i) in the cloudlet in cell g if service is placed in cell g or (ii) in the cloud in cell g if no service is placed in cell g
$\tilde{p}_g$	Processing units in the cloud in cell g even though service is placed in cell g		

synchronizing local game states between the cloudlet and the global state in the cloud. The required data transfer between cloudlet and cloud is denoted as d_g. The amount of processing units required for this case in the cloud is reduced to $\tilde{p}_g < p_g$.

In the following, the cost models for the two options, using cloud resources or service placement on cloudlets, are introduced. The cost for providing the service on cloud resources ($\gamma_g = 0$) in cell g consists of two parts: the cost for data transfer and the cost for computing. First, the SP has to pay a cost factor $\alpha_{\mathrm{CP},g}^{\mathrm{SP}}$ for utilizing the IP's network for each of the $\tilde{d}_g$ data units that need to be transmitted from the BS to the cloud. Second, the SP has to pay a cost factor $\beta_{\mathrm{CP},g}^{\mathrm{SP}}$ to the cloud provider for the amount p_g of processing units required in the cloud, resulting in the following cost function:

$$\tilde{\kappa}_g^{\mathrm{SP}} = (1-\gamma_g)\Big[\alpha_{\mathrm{CP},g}^{\mathrm{SP}}\tilde{d}_g + \beta_{\mathrm{CP},g}^{\mathrm{SP}} p_g\Big] \tag{4.1}$$

Note that the SP does not pay any cost for utilizing bandwidth from the users' devices to the BS, since these transmissions need to be made regardless of whether cloudlets are used or not. Therefore, this factor is not considered in our cost model.

The cost for providing the service placed on a cloudlet ($\gamma_g = 1$) in cell g consists of three parts: the cost for data transfer, the cost for computing resources in the cloud, and the payment between SP and IP for the service placement. The cost for data transfer contains the cost factor $\alpha_{\mathrm{CP},g}^{\mathrm{SP}} \geq 0$ for every data unit d_g that is transferred between the cloudlet and

cloud. The cost factor $\beta^{\mathrm{SP}}_{\mathrm{CP},g} \geq 0$ denotes the cost of each processing unit in the cloud. The payment between SP and IP for the service placement on a cloudlet is denoted by $\kappa^{\mathrm{SP}}_{\mathrm{IP},g}$. $\kappa^{\mathrm{SP}}_{\mathrm{IP},g}$ consists of multiple parts, e.g., the cost incurred for placing a service initially on a cloudlet as well as the processing and data transfer to the cloudlet. The bargaining between the IP and the SP to agree on the payment $\kappa^{\mathrm{SP}}_{\mathrm{IP},g}$ is discussed in Sections 4.2.2 and 4.2.3. The resulting cost function κ^{SP}_{g} of the SP in cases where the service is placed on a cloudlet in cell g is

$$\kappa^{\mathrm{SP}}_{g} = \gamma_g \Big[\kappa^{\mathrm{SP}}_{\mathrm{IP},g} + \alpha^{\mathrm{SP}}_{\mathrm{CP},g} d_g + \beta^{\mathrm{SP}}_{\mathrm{CP},g} \tilde{p}_g \Big]. \tag{4.2}$$

The cost function of the SP for providing the service in the whole grid G consists of two parts: (i) the cost for cells where the service is placed on a cloudlet and (ii) the cost for cells running the service in the cloud. Overall, the cost function κ^{SP} of the SP for the whole grid G is as follows:

$$\kappa^{\mathrm{SP}} = \sum_{g \in G} \Big[\kappa^{\mathrm{SP}}_{g} + \tilde{\kappa}^{\mathrm{SP}}_{g} \Big]. \tag{4.3}$$

The SP aims to reduce its cost κ^{SP} by placing the service on suitable cloudlets [267]. Only cells where the cost for running the service on a cloudlet is lower than the cost for using the cloud should be used [51]. The SP's cost reduction is a result of the fact that less data has to be sent through the IP's network backend from the BS to the cloud, since the service request will be handled directly at the cloudlet. By optimized selection of the cells, the cost given by Equation (4.3) is lower than the cost that the SP has to pay to the cloud provider in the case without any cloudlet due to less network usage in the IP's backend.

Infrastructure Provider

Cloudlets contribute to the IP's goal of reducing its cost [267] by reducing the network traffic in its network backend, since large parts of the service usage are handled directly by the cloudlet. However, the parameters of the IP's cost function differ from the parameters of the SP's cost function. While the SP has to pay the cloud provider or the IP, depending on using the cloud or cloudlets, the IP has to pay for establishing and maintaining the network. Cloudlets reduce this cost by adding computing and storage resources at the edge network and reducing the traffic in the core network, which leads to a cost reduction for the IP. To reduce as much cost as possible, a cloudlet should serve many users, otherwise the deployment cost could be higher than the cost reduction from the decreased network usage.

We distinguish between three components in the cost of the IP: fixed cost required for operating the infrastructure for deployment and maintenance of the service, cost for processing on the cloudlet, and the cost for transferring data over the IP's backend network.

The fixed cost ϕ^{IP}_{g} of the IP related to cell g contains all the infrastructure cost associated with deployment and maintenance of a service, which is assumed to be independent of the number of service requests in cell g. The cost for deploying the service, i.e., transferring the required data from the cloud to the cloudlet over the backend, only has to be paid if the service is not already deployed on the cloudlet in cell g. We do not consider service migration between different cloudlets, but we assume that a service is always offloaded from the cloud

to the cloudlet. The maintenance of a service, e.g., regular updates and monitoring, incurs a cost for processing and data transfer for the IP (although less compared to cells where the service is not deployed), even if the service is already deployed on the cloudlet in cell g.

The second component of the cost, the cost for processing on the cloudlet, is modeled in each cell g by the cost factor β_g^{IP} per processing unit. The total cost for processing is given by the cost factor β_g^{IP} multiplied by the number p_g of processing units required on the cloudlet in the given cell g.

The third component of the cost is the cost for transferring data over the IP's backend network. For each data unit transmitted over the backend network, a cost factor for α_g^{IP} is incurred. The total cost for data transfer is given by the number of data units d_g multiplied with the cost factor α_g^{IP}.

To make service placement on cloudlets profitable for the IP, the IP has to receive a payment by the SP, which is larger than the cost. The SP pays $\kappa_{\mathrm{IP},g}^{\mathrm{SP}}$ to the IP for placing the service on the cloudlet in cell g. How the payment $\kappa_{\mathrm{IP},g}^{\mathrm{SP}}$ is determined such that is acceptable for both the IP and the SP is the central problem discussed in this section.

Combining all the components of the cost and the payment of the SP, results in the following cost for the IP if a cloudlet is used in $g \in G$:

$$\kappa_g^{\mathrm{IP}} = \gamma_g \Big[\phi_g^{\mathrm{IP}} + \beta_g^{\mathrm{IP}} p_g + \alpha_g^{\mathrm{IP}} d_g - \kappa_{\mathrm{IP},g}^{\mathrm{SP}} \Big] \tag{4.4}$$

If no cloudlet is used in cell g (i.e., $\gamma_g = 0$), the IP only has the cost factor α_g^{IP} for utilizing the backend network to transfer each of the $\tilde{d}_g$ data units from the cloudlet to the cloud, leading to the following cost function:

$$\tilde{\kappa}_g^{\mathrm{IP}} = (1 - \gamma_g) \Big[\alpha_g^{\mathrm{IP}} \tilde{d}_g \Big] \tag{4.5}$$

This results in the following cost for the IP for the grid G:

$$\kappa^{\mathrm{IP}} = \sum_{g \in G} \Big[\kappa_g^{\mathrm{IP}} + \tilde{\kappa}_g^{\mathrm{IP}} \Big] \tag{4.6}$$

Note that the IP does not have to pay the cost given in Equation (4.6) directly to any stakeholder, but indirectly, e.g., through higher investments into the network to handle the amount of data from all services, which can be summarized as operating expenses. The cost for deploying a cloudlet and letting SPs place services on it will be paid by the SP to the IP to some degree, which results in a cost reduction.

Table 4.1 summarizes the mathematical notation and gives an overview of the used symbols.

Resource Limitations and Competition

We assume that sufficient resources are available at each cloudlet to serve the SP's demand for processing units p_g in cell g. In cases where the computational demand of the service would exceed the resources of the cloudlet, we assume that ϕ_g^{IP} (i.e., the cost for placing a service on a cloudlet incurred to the IP due to the initial data transfer) contains additional cost to scale the provided hardware in cell g accordingly.

This implies that although either a single SP may offer multiple services or multiple SPs may exist, our approach presented in Section 4.2.3 can be applied to each service separately, since the IP can handle unlimited services without causing conflict situations between SPs. Thus, we do not consider multiple SPs explicitly in the remainder of this section.

Finally, each cell g may have cloudlets from different IPs. However, we consider only a single IP in the remainder of the section without loss of generality, since our approach is applicable to each IP in a cell separately to find the IP with the highest cost reduction per cell. If multiple cloudlets from different IPs are available in one cell, the cost function applies to each IP. Furthermore, the SP may also opt to place the service on cloudlets of different IPs in one cell if the calculated cost reduction exceeds the cost in the cloud for that cell.

Problem Formulation

For each cell $g \in G$, the SP and IP need to come to an agreement whether the service should be placed on the cloudlet in cell g (i.e., $\gamma_g = 1$) or not (i.e., $\gamma_g = 0$). In the first case, they also need to agree on a cost $\kappa^{\text{SP}}_{\text{IP},g}$ to be paid by the SP to the IP. If they disagree, no service is placed in cell g (i.e., $\gamma_g = 0$, $\kappa^{\text{SP}}_{\text{IP},g} = 0$). To reach an agreement, a bargaining solution is needed that satisfies the following properties: (i) no participant of the bargaining should have disadvantages by participating in the bargaining (individual rationality) and (ii) the cost reduction caused by a service placement on cloudlets should be shared equally between the IP and SP (fairness). This gives a strong incentive for both the SP and the IP to participate in a bargaining for service placement. In the following, we derive a solution for service placement and the associated cost.

4.2.2 Bargaining with Complete Information

In this section, we present our solution to the problem stated in Section 4.2.1 under the assumption that both the IP and SP know all parameters of their own cost function and the cost function of the other stakeholder. In this case, a Nash Bargaining Solution (NBS) can be used to find an optimal agreement between IP and SP.

Nash bargaining is a two-person bargaining framework [177], where two stakeholders either reach an agreement a from a set A of alternatives or fail to reach an agreement, in which case the bargaining ends at a disagreement point d. Each stakeholder $i \in \{1, 2\}$ has a utility function U_i over the set of agreements and the disagreement point. Nash showed that under mild technical conditions, there exists a unique bargaining solution, called Nash Bargaining Solution (NBS), which satisfies a set of four axioms that any plausible bargaining solution should satisfy [177]. It can be shown [93, 177, 225] that an agreement $a \in A$ is a NBS if it solves the following optimization problem:

$$\begin{aligned} \max_{a \in A} \quad & (U_1(a) - U_1(d)) \cdot (U_2(a) - U_2(d)) \\ \text{s.t.} \quad & U_1(a) \geq U_1(d), U_2(a) \geq U_2(d). \end{aligned} \tag{4.7}$$

Hence, the NBS maximizes the product of both stakeholders' utility gains compared to the disagreement outcome. In the following, for each fixed cell g, we compute the NBS for the problem stated in Section 4.2.1.

Agreement Set and Disagreement Point

The set A_g of possible agreements between the IP and the SP in cell g is defined by

$$A_g := \left\{ \left(\gamma_g, \kappa^{\mathrm{SP}}_{\mathrm{IP},g}\right) | \gamma_g = 1, \kappa^{\mathrm{SP}}_{\mathrm{IP},g} \in (-\infty, \infty) \right\} \tag{4.8}$$

Note that the price $\kappa^{\mathrm{SP}}_{\mathrm{IP},g}$ may also be negative, since if placing the service in the cell is more profitable for the IP than the SP, the IP may have to pay the SP to reach an agreement.

If IP and SP do not come to an agreement for cell g, no service is placed in cell g. A disagreement in cell g is described by the following disagreement point:

$$\left(\gamma_g, \kappa^{\mathrm{SP}}_{\mathrm{IP},g}\right) = (0, 0) \tag{4.9}$$

Utilities of SP and IP

In Nash bargaining, each stakeholder has a utility function over the agreement set and the disagreement point. In our case, the utility functions per cell correspond to the negative cost functions per cell. Hence, according to Equation (4.1) and Equation (4.2), the utility of the SP for cell g is given by

$$\begin{aligned} U^{\mathrm{SP}}_g \left(\gamma_g, \kappa^{\mathrm{SP}}_{\mathrm{IP},g}\right) :=& \gamma_g \Big[- \kappa^{\mathrm{SP}}_{\mathrm{IP},g} - \alpha^{\mathrm{SP}}_{\mathrm{CP},g} d_g - \beta^{\mathrm{SP}}_{\mathrm{CP},g} \tilde{p}_g \Big] \\ &+ (1 - \gamma_g) \Big[- \alpha^{\mathrm{SP}}_{\mathrm{CP},g} \tilde{d}_g - \beta^{\mathrm{SP}}_{\mathrm{CP},g} p_g \Big] \end{aligned} \tag{4.10}$$

and, according to Equations (4.4) and (4.5), the utility of the IP for cell g is given by

$$\begin{aligned} U^{\mathrm{IP}}_g \left(\gamma_g, \kappa^{\mathrm{SP}}_{\mathrm{IP},g}\right) :=& \gamma_g \Big[- \phi^{\mathrm{IP}}_g - \alpha^{\mathrm{IP}}_g d_g - \beta^{\mathrm{IP}}_g p_g + \kappa^{\mathrm{SP}}_{\mathrm{IP},g} \Big] \\ &+ (1 - \gamma_g) \Big[- \alpha^{\mathrm{IP}}_g \tilde{d}_g \Big]. \end{aligned} \tag{4.11}$$

Disagreement Outcome

The disagreement outcome is given by the utilities of the SP and IP, if they choose the disagreement point:

$$\begin{aligned} &\left(U^{\mathrm{SP}}_g(0,0), U^{\mathrm{IP}}_g(0,0)\right) \\ &= \left(-\alpha^{\mathrm{SP}}_{\mathrm{CP},g} \tilde{d}_g - \beta^{\mathrm{SP}}_{\mathrm{CP},g} p_g, -\alpha^{\mathrm{IP}}_g \tilde{d}_g \right) \end{aligned} \tag{4.12}$$

Feasible Agreement Points

The IP and SP will only accept an agreement $\left(1, \kappa^{\mathrm{SP}}_{\mathrm{IP},g}\right) \in A_g$ if, for both of them, the agreement is better than the disagreement outcome.

An agreement $\left(1, \kappa_{\mathrm{IP},g}^{\mathrm{SP}}\right) \in A_g$ is better for the SP than the disagreement outcome, if $U_g^{\mathrm{SP}}\left(1, \kappa_{\mathrm{IP},g}^{\mathrm{SP}}\right) \geq U_g^{\mathrm{SP}}(0,0)$ holds, which by Equation (4.10) and Equation (4.12) is equivalent to

$$\kappa_{\mathrm{IP},g}^{\mathrm{SP}} \leq \alpha_{\mathrm{CP},g}^{\mathrm{SP}}(\tilde{d}_g - d_g) + \beta_{\mathrm{CP},g}^{\mathrm{SP}}(p_g - \tilde{p}_g) =: L_g^{\mathrm{SP}} \tag{4.13}$$

Hence, $L_g^{\mathrm{SP}} \geq 0$ gives an upper limit on the price that the SP would accept. The value L_g^{SP} corresponds to the cost reduction for the SP when a service is placed in cell g.

An agreement $\left(1, \kappa_{\mathrm{IP},g}^{\mathrm{SP}}\right) \in A_g$ is better for the IP than the disagreement outcome, if $U_g^{\mathrm{IP}}\left(1, \kappa_{\mathrm{IP},g}^{\mathrm{SP}}\right) \geq U_g^{\mathrm{IP}}(0,0)$ holds, which by Equation (4.11) and Equation (4.12) is equivalent to

$$\kappa_{\mathrm{IP},g}^{\mathrm{SP}} \geq \phi_g^{\mathrm{IP}} + \alpha_g^{\mathrm{IP}}(d_g - \tilde{d}_g) + \beta_g^{\mathrm{IP}} p_g =: l_g^{\mathrm{IP}} \tag{4.14}$$

Hence, l_g^{IP} gives a lower limit on the cost to be paid by the SP such that the IP would accept the agreement. The value l_g^{IP} corresponds to the cost increase that the IP experiences when the service is placed in cell g. Note that l_g^{IP} can also be negative ($l_g^{\mathrm{IP}} < 0$) if the IP has a cost reduction by placing the service in cell g even without a payment by the SP.

A feasible agreement point is an agreement $\left(1, \kappa_{\mathrm{IP},g}^{\mathrm{SP}}\right) \in A_g$ that satisfies Equation (4.13) and Equation (4.14). Such feasible agreement points do not necessarily exist. In detail, if $L_g^{\mathrm{SP}} < l_g^{\mathrm{IP}}$ holds, i.e., the SP's upper cost limit L_g^{SP} is lower than the IP's lower cost limit l_g^{IP}, by Equation (4.13) and Equation (4.14), no feasible agreement point exists. In this case, the IP and SP will choose the disagreement option $\gamma_g = 0$.

However, if $L_g^{\mathrm{SP}} \geq l_g^{\mathrm{IP}}$, each cost $\kappa_{\mathrm{IP},g}^{\mathrm{SP}}$ in $\left[l_g^{\mathrm{IP}}, L_g^{\mathrm{SP}}\right]$ leads to a feasible agreement.

Nash Bargaining Solution

We now formulate the optimization problem according to Equation (4.7) to compute the NBS. If feasible agreement points exist, i.e., if $L_g^{\mathrm{SP}} \geq l_g^{\mathrm{IP}}$ holds, the NBS is the optimal solution of the following problem:

$$\begin{aligned} \max_{\kappa_{\mathrm{IP},g}^{\mathrm{SP}}} \quad & f(\kappa_{\mathrm{IP},g}^{\mathrm{SP}}) \\ \text{s.t.} \quad & \kappa_{\mathrm{IP},g}^{\mathrm{SP}} \in [l_g^{\mathrm{IP}}, L_g^{\mathrm{SP}}] \end{aligned} \tag{4.15}$$

where

$$\begin{aligned} f(\kappa_{\mathrm{IP},g}^{\mathrm{SP}}) &:= \left(U_g^{\mathrm{SP}}\left(1, \kappa_{\mathrm{IP},g}^{\mathrm{SP}}\right) - U_g^{\mathrm{SP}}(0,0)\right) \\ &\quad \cdot \left(U_g^{\mathrm{IP}}\left(1, \kappa_{\mathrm{IP},g}^{\mathrm{SP}}\right) - U_g^{\mathrm{IP}}(0,0)\right) \\ &= -\left(\kappa_{\mathrm{IP},g}^{\mathrm{SP}}\right)^2 + \kappa_{\mathrm{IP},g}^{\mathrm{SP}}(l_g^{\mathrm{IP}} + L_g^{\mathrm{SP}}) - l_g^{\mathrm{IP}} L_g^{\mathrm{SP}} \end{aligned} \tag{4.16}$$

To compute the NBS, we set the derivative to zero, i.e.,

$$0 = f'(\kappa_{\mathrm{IP},g,NBS}^{\mathrm{SP}}) = -2\kappa_{\mathrm{IP},g,NBS}^{\mathrm{SP}} + l_g^{\mathrm{IP}} + L_g^{\mathrm{SP}}, \tag{4.17}$$

we get

$$\kappa^{\mathrm{SP}}_{\mathrm{IP},g,NBS} = \frac{1}{2}\left(l^{\mathrm{IP}}_g + L^{\mathrm{SP}}_g\right) \tag{4.18}$$

and since $f''(\kappa^{\mathrm{SP}}_{\mathrm{IP},g,NBS}) = -2 < 0$, and $\kappa^{\mathrm{SP}}_{\mathrm{IP},g,NBS} \in [l^{\mathrm{IP}}_g, L^{\mathrm{SP}}_g]$, the result in Equation (4.18) gives the optimal solution of Equation (4.15). Hence, if $L^{\mathrm{SP}}_g \geq l^{\mathrm{IP}}_g$ holds, according to the NBS, SP and IP will agree on the price $\kappa^{\mathrm{SP}}_{\mathrm{IP},g,NBS}$ in Equation (4.18), which is the average of L^{SP}_g and l^{IP}_g. Intuitively, SP and IP equal out their cost so that after the payment, both benefit from the same cost reduction.

4.2.3 Iterative Bargaining with Incomplete Information

In Section 4.2.2, we have presented the NBS solution for the case that the SP and the IP have complete information about their own cost function and the cost function of the bargaining partner. In this section, we consider two types of incomplete information in the system model: incomplete information about the cost factors of the bargaining partner and incomplete information about the service usage. Furthermore, we propose a novel iterative bargaining approach under incomplete information to overcome the challenges imposed by the incomplete information.

Nash Bargaining with Incomplete Information

We extend the Nash bargaining problem from Section 4.2.2 to incorporate incomplete information. The first type of incomplete information is concerning the cost factors of the bargaining partner. The IP does not know the cost factor $\alpha^{\mathrm{SP}}_{\mathrm{CP},g}$ for transferring data in the cloud and the cost factor $\beta^{\mathrm{SP}}_{\mathrm{CP},g}$ for processing in the cloud. The SP does not know the cost ϕ^{IP}_g for placing a service, the cost factor α^{IP}_g for transferring data over the network and the cost factor β^{IP}_g for processing on the cloudlet. This type of information is relevant to predict the potential cost savings of the bargaining partner in case of an agreement. The SP and IP typically do not want to share this private information with each other, therefore we need to consider this information as incomplete in the bargaining procedure.

The second type of incomplete information is concerning the service usage in the cell g. The IP has no information about the SP's users activity and situation. Therefore, the prediction of service usage from the IP's perspective can only be based on other information. The service usage directly affects the cost function κ^{IP} of the IP. In particular, the change l^{IP}_g in the cost function by a service placement is not known by the IP.

To incorporate incomplete information about the service usage we present two different approaches: No Information Sharing (NIS) uses individual service usage predictions by the IP and the SP, and Partial Information Sharing (PIS), where the SP shares its service usage predictions with the IP.

Pre-Bargaining Information Acquisition and Sharing

Cost Factor Predictions Before the bargaining procedure starts, the IP and the SP acquire information about the bargaining partner. It is possible to obtain predictions of the bargaining partner's cost factors within a range of possible values from publicly available sources, e.g., from price lists of competitors. Although some information about the bargaining partner can be obtained using publicly available sources, the available information on both sides is still incomplete, hence IP and SP cannot simply apply the NBS from Section 4.2.2.

The SP has to predict l_g^{IP} (Equation (4.14)), which is the minimum price the IP would accept for placing the service on the cloudlet in cell g. We model the incomplete information as the uncertainty of the SP in l_g^{IP}. For this uncertainty from the perspective of the SP, we introduce a probability density function (PDF) of l_g^{IP}, which will be denoted as $p_{l_g^{\mathrm{IP}}}(l_g^{\mathrm{IP}})$. We assume no further knowledge about the IP exists, leading to the assumption of a uniform distribution between a lower bound $l_{g,n,\min}^{\mathrm{IP}}$ and an upper bound $l_{g,n,\max}^{\mathrm{IP}}$, resulting in the PDF

$$p_{l_g^{\mathrm{IP}}}(l_g^{\mathrm{IP}}) = \begin{cases} \frac{1}{l_{g,n,\max}^{\mathrm{IP}} - l_{g,n,\min}^{\mathrm{IP}}}, & l_g^{\mathrm{IP}} \in [l_{g,n,\min}^{\mathrm{IP}}, l_{g,n,\max}^{\mathrm{IP}}] \\ 0, & \text{else} \end{cases}. \tag{4.19}$$

The SP can obtain a lower bound $l_{g,n,\min}^{\mathrm{IP}}$, e.g., by predicting the cost for the additional hardware and energy of the IP for service placement [135], and an upper bound $l_{g,n,\max}^{\mathrm{IP}}$, e.g., by using public price lists for service placement [13].

The IP has to predict L_g^{SP} (Equation (4.13)), which is the upper limit on the price that the SP would accept. Analogously to the cost factor prediction of the SP, we introduce for this uncertainty of the IP a PDF of L_g^{SP}, which will be denoted as $p_{L_g^{\mathrm{SP}}}(L_g^{\mathrm{SP}})$. Furthermore, we assume that the IP can determine an interval for L_g^{SP} described by a lower bound $L_{g,n,\min}^{\mathrm{SP}}$ and an upper bound $L_{g,n,\max}^{\mathrm{SP}}$, resulting in the PDF

$$p_{L_g^{\mathrm{SP}}}(L_g^{\mathrm{SP}}) = \begin{cases} \frac{1}{L_{g,n,\max}^{\mathrm{SP}} - L_{g,n,\min}^{\mathrm{SP}}}, & L_g^{\mathrm{SP}} \in [L_{g,n,\min}^{\mathrm{SP}}, L_{g,n,\max}^{\mathrm{SP}}] \\ 0, & \text{else} \end{cases}. \tag{4.20}$$

The IP can obtain the lower bound $L_{g,n,\min}^{\mathrm{SP}}$ by predicting the reduction in cloud and backhaul cost of the SP (e.g., [163]), and the upper bound $L_{g,n,\max}^{\mathrm{SP}}$ by using public price lists of cloud providers (e.g., [12], [100]).

In the next step, the IP and the SP predict the service usage in each cell g of the grid G.

Service Usage Prediction The SP can predict the service usage based on measurements of the users' activity, users' context and general information like weather conditions and population densities. The exact service prediction procedure of the SP will not be further discussed in this approach, since there are several proposals in the literature (e.g., [81, 142]). We model the result of the SP's prediction of the service processing requirements p_g and data d_g as

$$\hat{p}_g^{\mathsf{SP}} = p_g + n_g^{\mathsf{p,SP}}, \quad \hat{d}_g^{\mathsf{SP}} = d_g + n_g^{\mathsf{d,SP}} \tag{4.21}$$

where $n_g^{\mathsf{p,SP}}$ ($n_g^{\mathsf{d,SP}}$) is a Gaussian distributed random variable with a standard deviation of $\sigma_g^{\mathsf{p,SP}}$ ($\sigma_g^{\mathsf{d,SP}}$). The accuracy of the prediction, which is given by $\sigma_g^{\mathsf{p,SP}}$ and $\sigma_g^{\mathsf{d,SP}}$, depends on the cell g. The remaining data $\tilde{d}_g$ and service processing requests $\tilde{p}_g$ to the cloud are predicted analogously.

The IP has no information about the SP's users and therefore relies on situational information in each cell g, such as weather conditions and population densities. We model the IP's prediction of the service processing requirements p_g and data d_g as

$$\hat{p}_g^{\mathsf{IP}} = p_g + n_g^{\mathsf{p,IP}}, \quad \hat{d}_g^{\mathsf{IP}} = d_g + n_g^{\mathsf{d,IP}} \tag{4.22}$$

where $n_g^{\mathsf{p,IP}}$ ($n_g^{\mathsf{d,IP}}$) is a Gaussian distributed random variable with a standard deviation of $\sigma_g^{\mathsf{p,IP}}$ ($\sigma_g^{\mathsf{d,IP}}$). Note that $\sigma_g^{\mathsf{p,IP}}$ and $\sigma_g^{\mathsf{d,IP}}$ may be substantially larger than $\sigma_g^{\mathsf{p,SP}}$ and $\sigma_g^{\mathsf{d,SP}}$ respectively, since the IP has no information about the SP's users. The prediction of the remaining data $\tilde{d}_g$ and service processing requests $\tilde{p}_g$ to the cloud are modelled analogously to Equation (4.22).

In the next section, the sharing of the service usage predictions is discussed. For this procedure, two alternatives will be presented. The first one is without any information sharing between the stakeholders, whereas in the second one both stakeholders share their service usage prediction with each other.

No Information Sharing (NIS) This approach does not involve any information sharing between IP and SP before the bargaining procedure. Therefore, both stakeholders use a different service usage prediction to forecast their resulting cost of a service placement. To predict the IP's cost l_g^{IP} in case of a service placement in cell g, the IP considers the service usage prediction using Equation (4.22). Since the IP knows its own cost factors, the IP can calculate l_g^{IP} by Equation (4.14). The SP can predict its potential cost savings L_g^{SP} (Equation (4.13)) in case of a service placement with its service usage prediction according to Equation (4.21).

Partial Information Sharing (PIS) In this approach, both stakeholders report their service usage predictions for cell g to the bargaining partner. For cases where the SP has no experience with deploying the service in cell g, the prediction of the IP could be more accurate. In cells with high service usage in the past, the prediction of the SP could be more accurate. Sharing the service usage predictions does not introduce negative consequences for both, since both can measure the service usage anyway. Both the IP and SP will use the more accurate prediction to calculate their own cost functions $l_g^{\mathrm{IP}}, L_g^{\mathrm{SP}}$.

Note that using the PIS variant for the upcoming iterative bargaining approach, one of the stakeholders, or both, could lie about the predicted service usage, whereas using the NIS variant, this is not possible. To prevent stakeholders from abusing wrong predictions, there are several mechanisms to enforce truthfulness in repeated games, e.g., reputation-based methods [262] we will not further investigate it in this thesis. The remaining structure of our iterative bargaining approach is the same, regardless of whether NIS or PIS is used to acquire all required information and compute the IP's and SP's cost functions.

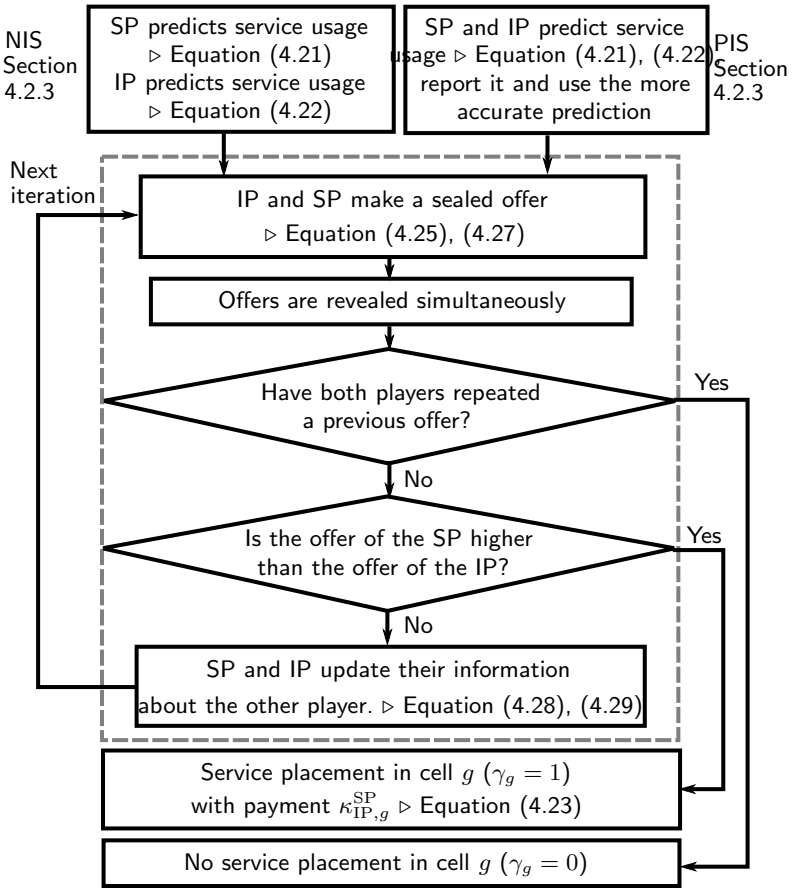


Figure 4.3: The iterative bargaining procedure

Using either its own prediction of d_g, $\tilde{d}_g$ and p_g or the values it got from the SP, the IP can predict the minimum price l_g^{IP} that should be paid by the SP. However, the IP has limited knowledge about the valuation L_g^{SP}, i.e., how profitable a service placement is for the SP. The SP, on the other hand, computes the maximum cost L_g^{SP} that it would accept for using the cloudlet. However, the SP has limited knowledge about the valuation l_g^{IP}, i.e., how profitable or expensive a service placement on a cloudlet in cell g is for the IP.

Iterative Bargaining

Our solution is an iterative bargaining approach with sealed offers, as shown in Figure 4.3. The value n indicates the index of the current iteration.

Offering Phase The SP offers a maximum acceptable cost $o_{g,n}^{\mathrm{SP}}$ to pay for using the cloudlet, whereas the IP offers a minimum acceptable cost $o_{g,n}^{\mathrm{IP}}$ for using the cloudlet. The SP and the IP update their offers according to their corresponding information in round n, i.e., the cost $o_{g,n}^{\mathrm{SP}}$ and $o_{g,n}^{\mathrm{IP}}$ may not correspond to their actual true valuations L_g^{SP} and l_g^{IP}.

Revealing Phase Both offers are revealed simultaneously. If the SP's offer is higher than the offer of the IP $o_{g,n}^{\mathrm{SP}} > o_{g,n}^{\mathrm{IP}}$ the bargaining is finished. The final cost is calculated as

$$\kappa_{\mathrm{IP},g}^{\mathrm{SP}} = \frac{1}{2} \cdot (o_{g,n}^{\mathrm{SP}} + o_{g,n}^{\mathrm{IP}}). \tag{4.23}$$

Choosing the final cost as the average of the two offers has been shown to maximize the social cost [176]. If $o_{g,n}^{\mathrm{SP}} < o_{g,n}^{\mathrm{IP}}$ the bargaining will continue, since the SP's offer is lower than the minimum acceptable cost of the IP. If both repeat a previous offer, i.e. $o_{g,n}^{\mathrm{SP}} = o_{g,n-1}^{\mathrm{SP}}$, and $o_{g,n}^{\mathrm{IP}} = o_{g,n-1}^{\mathrm{IP}}$, neither SP nor IP are willing to give an offer closer to the acceptable region. They will disagree, the bargaining procedure will stop, and the service will not be placed in this cell. Furthermore, we set a limit N for the number of iterations, such that the disagreement outcome is chosen if $n > N$.

Update Phase After seeing the offer of the other stakeholder, both stakeholders update their knowledge about the valuation of the other stakeholder. Then, the next round $n+1$ of the bargaining process starts.

SP and IP only have access to incomplete information regarding the valuation of the other stakeholder and consequently do not know the worst case of the other stakeholder. Therefore, we derive an optimal bidding strategy for SP and IP under incomplete information. The expected profit of the SP considering the incomplete information about l_g^{IP} depending on its offer $o_{g,n}^{\mathrm{SP}}$ in cell g is

$$\pi_{g,\mathrm{SP}}(o_{g,n}^{\mathrm{SP}}) = \mathbb{E}_{l_g^{\mathrm{IP}}}(L_g^{\mathrm{SP}} - \kappa_{\mathrm{IP},g}^{\mathrm{SP}}) = \tag{4.24}$$
$$\int_{-\infty}^{o_{g,n}^{\mathrm{SP}}} \left(L_g^{\mathrm{SP}} - \frac{1}{2}\left(o_{g,n}^{\mathrm{SP}} + s \right) \right) \cdot p_{l_g^{\mathrm{IP}}}(s)\,\mathrm{d}s.$$

Three edge cases for $o_{g,n}^{\mathrm{SP}}$ can be distinguished: (i) $o_{g,n}^{\mathrm{SP}} < l_{g,n,\min}^{\mathrm{IP}}$: the offer of the SP is smaller than the predicted lower bound for the cost of the IP, i.e., the expected profit (Equation (4.24)) is zero, since the probability of an agreement is zero; (ii) $o_{g,n}^{\mathrm{SP}} > L_g^{\mathrm{SP}}$: the offer of the SP is higher than its own benefit from an agreement, i.e., the expected profit (Equation (4.24)) may be negative; (iii) $o_{g,n}^{\mathrm{SP}} > l_{g,n,\max}^{\mathrm{IP}}$: the SP's offer is higher than the maximum predicted cost $l_{g,n,\max}^{\mathrm{IP}}$ of the IP, i.e., this offer leads to the probability of an agreement of 1.0, but is clearly suboptimal for the expected profit of the SP. The SP could decrease its offer and increase its expected profit.

We assume that SP and IP are risk-neutral, i.e., that they maximize their individual expected utility. Furthermore, we assume individual rationality, i.e., a stakeholder only gives an offer if the expected utility is positive. The optimal offer $o_{g,n}^{\mathrm{SP}}$ for the SP in g is

$$\begin{aligned} o_{g,\mathrm{SP}}^{*,(t)} &= \max_{o_{g,n}^{\mathrm{SP}}} \quad \pi_{g,\mathrm{SP}}(o_{g,n}^{\mathrm{SP}}) \\ &= \begin{cases} \frac{2}{3} \cdot (L_g^{\mathrm{SP}} + \frac{1}{2} l_{g,n,\min}^{\mathrm{IP}}) & L_g^{\mathrm{SP}} < \frac{3}{2} l_{g,n,\min}^{\mathrm{IP}} \\ L_g^{\mathrm{SP}}, & L_g^{\mathrm{SP}} > \frac{3}{2} l_{g,n,\min}^{\mathrm{IP}} \end{cases} \end{aligned} \tag{4.25}$$

The expected profit of the IP considering the incomplete information about L_g^{SP} depending on its offer $o_{g,n}^{\mathrm{IP}}$ in cell g is

$$\pi_{g,\mathrm{IP}}(o_{g,n}^{\mathrm{IP}}) = \mathbb{E}_{L_g^{\mathrm{SP}}}(\kappa_{\mathrm{IP},g}^{\mathrm{SP}} - l_g^{\mathrm{IP}}) = \int_{o_{g,n}^{\mathrm{IP}}}^{\infty} \left(\frac{1}{2}\left(o_{g,n}^{\mathrm{SP}} + s\right) - l_g^{\mathrm{IP}}\right) \cdot p_{L_g^{\mathrm{SP}}}(s)\,\mathrm{d}s. \tag{4.26}$$

Three edge cases for $o_{g,n}^{\mathrm{IP}}$ can be distinguished: (i) $o_{g,n}^{\mathrm{IP}} < l_g^{\mathrm{IP}}$: the offer of the IP is smaller than its own cost induced by the service placement, i.e., the expected profit (Equation (4.26)) may be negative; (ii) $o_{g,n}^{\mathrm{IP}} < L_{g,n,\min}^{\mathrm{SP}}$: in this case, the probability of an agreement is 1.0, since the IP offers less than the minimum predicted cost reduction of the SP, i.e., this type of offer is suboptimal, since the IP could increase $o_{g,n}^{\mathrm{IP}}$ and increase its expected profit; (iii) $o_{g,n}^{\mathrm{IP}} > L_{g,n,\max}^{\mathrm{SP}}$: the offer of the IP is higher than the expected cost reduction at the SP, i.e., the expected profit (Equation (4.26)) is zero, since the probability of an agreement is zero. The optimal offer $o_{g,n}^{\mathrm{SP}}$ for the IP in cell g is

$$\begin{aligned} o_{g,\mathrm{IP}}^{*,(t)} &= \max_{o_{g,n}^{\mathrm{SP}}} \quad \pi_{g,\mathrm{SP}}(o_{g,n}^{\mathrm{SP}}) \\ &= \begin{cases} l_g^{\mathrm{IP}}, & l_g^{\mathrm{IP}} > \frac{3}{2} L_{g,n,\max}^{\mathrm{SP}} \\ \frac{2}{3} \cdot (l_g^{\mathrm{IP}} + \frac{1}{2} L_{g,n,\max}^{\mathrm{SP}}) & l_g^{\mathrm{IP}} < \frac{3}{2} L_{g,n,\max}^{\mathrm{SP}} \end{cases}. \end{aligned} \tag{4.27}$$

After seeing the offers of the other stakeholder, both decide to update their information about the valuation of the other stakeholder. For this update, we choose a sequential linear estimator with the adjustment rates λ^{SP}, λ^{IP} based on the observation of the offer of the other stakeholder:

$$l_{g,n+1,\min}^{\mathrm{IP}} = l_{g,n,\min}^{\mathrm{IP}} + \lambda^{\mathrm{SP}} \cdot (o_{g,n}^{\mathrm{SP}} - l_{g,n,\min}^{\mathrm{IP}}) \tag{4.28}$$

$$L_{g,n+1,\max}^{\mathrm{SP}} = L_{g,n,\max}^{\mathrm{SP}} + \lambda^{\mathrm{IP}} \cdot (o_{g,n}^{\mathrm{IP}} - L_{g,n,\max}^{\mathrm{SP}}). \tag{4.29}$$

In case of complete information (both sides know the cost function of the other stakeholder), the proposed algorithm yields the NBS, as computed in Section 4.2.2, in the first iteration. The final cost is the same as in the NBS:

$$\begin{aligned} \kappa_{\mathrm{IP},g}^{\mathrm{SP}} &= \frac{1}{2} \cdot (o_{g,n}^{\mathrm{SP}} + o_{g,n}^{\mathrm{IP}}) \\ &= \frac{1}{2} \cdot \left(\frac{2}{3} \cdot \left(L_g^{\mathrm{SP}} + \frac{1}{2} l_g^{\mathrm{IP}}\right) + \frac{2}{3} \cdot \left(l_g^{\mathrm{IP}} + \frac{1}{2} L_g^{\mathrm{SP}}\right)\right) \\ &= \frac{1}{2} \left(l_g^{\mathrm{IP}} + L_g^{\mathrm{SP}}\right) = \kappa_{\mathrm{IP},g,NBS}^{\mathrm{SP}}. \end{aligned} \tag{4.30}$$

However, IP and SP typically do not exchange their cost functions and private valuations. This incomplete information can lead to inefficiencies compared to the NBS with complete information [79]. For every cell $g \in G$, the resulting κ_g^{SP} and κ_g^{IP} may not result in the minimum cost for both stakeholders, but if both agree on a cost, it will be lower than $\tilde{\kappa}_g^{\mathrm{SP}}$ and $\tilde{\kappa}_g^{\mathrm{IP}}$ would be. This entire procedure is repeated for every cell $g \in G$, resulting in κ^{SP} and κ^{IP}, respectively.

Complexity Analysis

To discuss the complexity of the proposed approach, the update parameters have to be discussed first, since they play a crucial role concerning how long the bargaining will take. Selection of the maximum iteration count N and the adjustment rates $\lambda^{\{\mathrm{IP,SP}\}}$ of the SP's and IP's offer are interdependent. A small adjustment rate $\lambda^{\{\mathrm{IP,SP}\}}$ leads to a many iterations until SP and IP either agree or finally disagree by repeating a previous offer. Choosing N too small will lead to more cells without an agreement, although it might be profitable for both to reach an agreement. A high value of $\lambda^{\{\mathrm{IP,SP}\}}$ is undesirable for the SP and IP, respectively, since the offer is corrected too much in favor of the other stakeholder.

We assume that both stakeholders agree on a maximum number of iterations, e.g., $N = 10$, due to time constraints, i.e., the bargaining should not take too much time compared to the duration of service placement. Each stakeholder can tune its adjustment rate $\lambda^{\{\mathrm{IP,SP}\}}$ individually, e.g., based on heuristics.

The bargaining procedure results in one of the two cases in each cell g: (i) the stakeholders agree on a cost or (ii) they disagree. In both cases, the number of iterations is bounded by N, since the bargaining is stopped after N rounds without agreement. The iterative bargaining is repeated for every cell $g \in G$. Thus, our iterative bargaining approach will converge at the latest after $N \cdot |G|$ iterations in the worst case. In the case of complete information, iterative bargaining yields the NBS, thus both stakeholders will agree on the payment stated in Equation (4.30) for a particular cloudlet during the first iteration, which yields $1 \cdot |G|$ iterations in the best case.

4.2.4 Experimental Evaluation

Case Study and Data Set

As discussed in the Introduction, cloudlets are especially suitable for processing latency-sensitive applications like mobile AR games. Therefore, our evaluation uses a real-world data set that we collected from Niantic's AR game Ingress, which was empirically investigated by Felka et al. [81] and is available on request for scientific purposes. In this game, players visit so-called portals that are linked to real points of interest and try to capture them, leading to continuous player movements in the real world. The data set contains the user's service requests including the location of a request, which will be called game *actions* in the following. One game action is one interaction between a player and a portal by either trying to capture or defend it.

The data set was collected over a time period starting on 1st of January 2016 until the 31st of May 2017 (i.e., almost 1.5 years), but we focus on the data collected for the year 2016. The data for this period is almost complete, except for a 5-day maintenance period (from 2016/09/20 to 2016/09/24). In the considered period of time, about $21,250$ users of the game have made over $17 \cdot 10^6$ game actions at $53,259$ portals in the observation area. The data set covers urban, metropolitan, and rural areas with a total area of $25{,}200\,\mathrm{km}^2$. We divided the area into cells of $1\,\mathrm{km} \times 1\,\mathrm{km}$ each, since 5G base stations approximately have a radius of about $500\,\mathrm{m}$, depending on the selected frequency and antenna. Subsequently, we assigned the game actions to the corresponding cells of the grid and summed all game

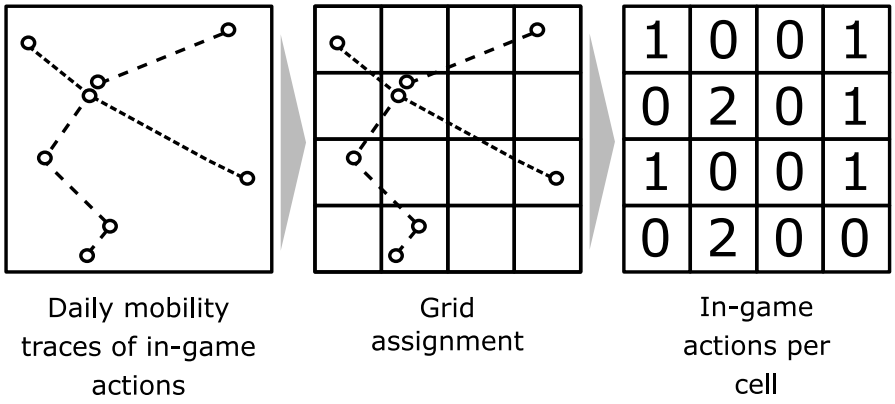


Figure 4.4: Data preprocessing for every day

Table 4.2: Mathematical notation and their values for the Take-it-or-Leave-it approach

Variable	Description	Value	Source
$\beta^{\mathrm{SP}}_{\mathrm{IP},g}$	Prcessing cost SP has to pay to IP	$unif[\beta^{\mathrm{IP}}_g, 4 \cdot \beta^{\mathrm{IP}}_g]$	[12, 13, 100]
$\phi^{\mathrm{SP}}_{\mathrm{IP},g}$	Service placement cost SP has to pay to IP	$unif[\phi^{\mathrm{IP}}_g, 4 \cdot \phi^{\mathrm{IP}}_g]$	[12, 13, 100]

actions on a daily basis within a cell to determine the total daily number of service requests of a cell. This process is visualized in Figure 4.4. In our case study, the SP deploys the parts of the game (i.e., the service) relevant for the spatial area of the grid to cloudlets, while the IP owns the infrastructure, represented by the $25,200$ cells. In our scenario, each cell represents a possible deployment point for the services. The data set contains the number of in-game actions for each cell, representing the geographical service usage of Ingress.

Experimental Setup

Besides our iterative bargaining approach with two alternatives (PIS and NIS), we also include two additional cost models: a traditional Take-it-or-Leave-it approach and the NBS.

Take-it-or-Leave-it The Take-it-or-Leave-it (ToL) approach represents a typical cost-based model. The IP independently selects prices in the cost functions as follows: First, placing a service in cell g incurs a fixed cost $\phi^{\mathrm{SP}}_{\mathrm{IP},g} \geq 0$ that the SP has to pay. This is the cost for placing the service on the cloudlet, i.e., transferring required data from cloud to the particular cloudlet or maintaining a service on a cloudlet. If in a previous bargaining the service was already placed on the cloudlet in cell g, ϕ^{IP}_g and thus $\phi^{\mathrm{SP}}_{\mathrm{IP},g}$ is only a fraction of the original cost, since the deployment cost is no longer included (the composition of ϕ^{IP}_g is explained at the end of Section 4.2.4 on page 67). Hence, the fixed cost $\phi^{\mathrm{SP}}_{\mathrm{IP},g}$ depends on the specific cell g. Second, the SP has to pay a cost factor $\beta^{\mathrm{SP}}_{\mathrm{IP},g} \geq 0$ to the IP for each of the p_g processing units required on the cloudlet in cell g:

$$\kappa^{\mathrm{SP}}_{\mathrm{IP},g} = \phi^{\mathrm{SP}}_{\mathrm{IP},g} + \beta^{\mathrm{SP}}_{\mathrm{IP},g} p_g \tag{4.31}$$

Table 4.2 summarizes the variables required for the ToL approach for this evaluation.

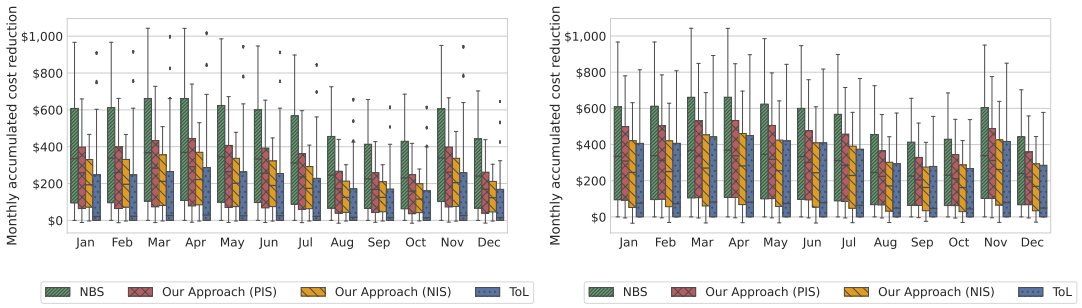
Table 4.3: Evaluation parameters

Variable	Description	Value	Source
$actions_g$	Number of game actions in cell g	Taken from data set	[81]
$\tilde{d}_g$	Data from BS to cloud if $\gamma_g = 1$	$actions_g \cdot DU$	[17]
d_g	Data from BS to cloud if $\gamma_g = 0$	$\tilde{d}_g \cdot (1 - 0.92)$	[253]
DU	Data units	$d \cdot$ MB, $d \in [1, 8]$	[17]
p_g	Processing of actions	$actions_g \cdot PU$	[12, 100]
$\tilde{p}_g$	Processing in cloud even if $\gamma_g = 1$	$p_g \cdot 0.1$	[253]
PU	Processing time	$[0.125\text{s}, 2.5\text{s}, 8.3\text{s}]$	[69, 149, 227]
$\alpha^{\text{SP}}_{\text{CP},g}$	Cost factor for data SP has to pay to cloud provider in cell g	0.00012	[12, 100]
α^{IP}_{g}	Cost factor for data incurred for IP in cell g	0.00006	[13]
$\beta^{\text{SP}}_{\text{CP},g}$	Cost factor for processing SP has to pay to cloud provider in cell g	0.000029	[12, 100]
β^{IP}_{g}	Cost factor for processing incurred for IP in cell g	0.000087	[13]
ϕ^{IP}_{g}	Cost for placing cloudlet in cell g incurred for the IP	$128\,\text{MB} \cdot \alpha^{\text{SP}}_{\text{CP},g}$	[12, 13, 100]
N	Number of bargaining iterations	10	Parameter discussion in Section 4.2.3
λ^{SP}	Adj. rate of the SP's offer during iter. bargaining	$[0.1, 0.2, 0.3]$	
λ^{IP}	Adj. rate of the IP's offer during iter. bargaining	$[0.1, 0.2, 0.3]$	
$\hat{\alpha}^{\text{IP}}_{g}$	Predicted cost factors from public sources	$unif[0.9, 1.1] * \alpha^{\text{IP}}_{g}$	Cost factor prediction in Section 4.2.3
$\hat{\alpha}^{\text{SP}}_{\text{CP},g}$		$unif[0.9, 1.1] * \alpha^{\text{SP}}_{\text{CP},g}$	
$\hat{\beta}^{\text{IP}}_{g}$		$unif[0.9, 1.1] * \beta^{\text{IP}}_{g}$	
$\hat{\beta}^{\text{SP}}_{\text{CP},g}$		$unif[0.9, 1.1] * \beta^{\text{SP}}_{\text{CP},g}$	
$\hat{\phi}^{\text{IP}}_{g}$		$unif[0.9, 1.1] * \phi^{\text{IP}}_{g}$	
$l^{\text{IP}}_{g,min}$	SP's prediction of the IP's lower limit	$\hat{\phi}^{\text{IP}}_{g} + \hat{\alpha}^{\text{IP}}_{g}(d_g - \tilde{d}_g) + \hat{\beta}^{\text{IP}}_{g} p_g$	
$L^{\text{SP}}_{g,max}$	IP's prediction of the SP's lower limit	$\hat{\alpha}^{\text{SP}}_{\text{CP},g}(\tilde{d}_g - d_g) + \hat{\beta}^{\text{SP}}_{\text{CP},g}(p_g - \tilde{p}_g)$	

To place a service on a cloudlet, the SP has to pay the price the IP asks for. The SP makes the placement decision. Only if the SP can reduce its cost by using a cloudlet, the SP performs a placement. This approach represents a lower-bound scenario in our evaluation, since the two stakeholders do not cooperate in any respect to achieve joint benefits.

Nash Bargaining The second approach represents the optimal solution, i.e., the case where both stakeholders cooperate, have complete information about each other's cost, and achieve minimal social cost according to Section 4.2.2.

Iterative Bargaining The third approach includes the two variants of our iterative bargaining approach. In the first variant, both stakeholders do not exchange any information. In contrast, in the second variant, information is partly shared between the stakeholders, e.g., the SP provides service usage predictions to the IP (see Section 4.2.3).



(a) Service provider **(b)** Infrastructure provider

Figure 4.5: Monthly accumulated cost reduction over one year for all approaches

The data set consists of in-game actions that reflect the service usage of the game but are not technical measures that we can directly use in our simulation. We therefore translate the in-game actions into technical measures such as the network traffic or the required processing power in a cell to conduct a meaningful simulation. In the following, we describe this process in more detail and present an overview of the parameters in Table 4.3.

Regarding network traffic, one hour of play in Ingress causes about 28 MB of traffic on a mobile device of one player, while a player performs an average of 14.56 game actions per hour. By dividing the network traffic of one-hour gameplay by the average number of game actions per hour, we get a value of 2 MB per game action. While 2 MB per game action seems like a lot, it includes all unrecorded activities, which also generate traffic (view the map or photos of the portals) and can thus generate traffic without performing any logged game action. Therefore, we consider an average of 2 MB per game action to be realistic. However, to avoid being specific for the game Ingress, we vary this parameter in our evaluation to match the network traffic of other popular AR games [17]. Thus, we assume that mobile AR games can generate data units DU ranging from 1 MB to 8 MB. If there is no placement of a service in cell g, we assume that the entire data has to be sent to the cloud, resulting in $\tilde{d}_g = actions_g \cdot DU$. Based on the results of a study by Wang et al. [253], we assume an average reduction of 92% in network traffic to the cloud and remaining overhead traffic of 8%, if there is a service placement on a cloudlet in a particular cell, thus $d_g = \tilde{d}_g \cdot (1 - 0.92)$.

As already explained for the data units, we also vary the processing units to represent AR games with different processing requirements, which is based on required CPU cycles. Based on the works of Al-Shuwaili and Simeone [227], Liu et al. [149] and Dinh et al. [69], applications processed on cloudlets or similar edge computing resources require between $3 \cdot 10^8$ CPU cycles for simple applications and $20 \cdot 10^{10}$ CPU cycles for long-running applications. Based on the performance of modern CPUs, we assume an average of 2.5 GHz processing speed, resulting in 0.125 s and 8.3 s execution time, respectively. As a third service type we introduce an execution time of 2.5 s, resulting in $PU = [0.125, 2.5, 8.3]$. For processing one game action in the cloud (i.e., no cloudlet is used), we define $p_g = actions_g \cdot PU$. Furthermore, according to Wang et al., $8\% - 12\%$ of requests need to be forwarded from the edge to the cloud server [253]. Therefore, we assume that 10% of all game actions require one additional processing unit in the cloud (e.g., to synchronize the game state), resulting in $\tilde{p}_g = p_g \cdot 0.1$.

As discussed in Section 4.2.3, each stakeholder can tune its adjustment rate $\lambda^{\{\mathrm{IP,SP}\}}$ individually. Therefore, we select three values for $\lambda^{\{\mathrm{IP,SP}\}}$ and simulate all combinations of these

three values between both stakeholders.

Now that the various service types have been defined, the following result emerges with regard to the number of simulations: for each combination of PU, DU, λ^{SP} and λ^{IP}, we simulate the four different bargaining approaches. Each simulation run simulates the placement behavior on a daily basis. This results in 311,040 simulations:

$$\begin{aligned} &|\text{Placement approaches}| \cdot |PU| \cdot |DU| \\ &\quad \cdot |\lambda^{\mathrm{IP}}| \cdot |\lambda^{\mathrm{SP}}| \cdot |days| = \\ &\quad 4 \cdot 3 \cdot 8 \cdot 3 \cdot 3 \cdot 360 = 311,040 \end{aligned} \tag{4.32}$$

Finally, we select and specify the parameters for the cost functions for the simulation. For data transmission cost factor to the cloud ($\alpha^{\mathrm{SP}}_{\mathrm{CP},g}$), we use public price lists of Amazon AWS Lambda [12] and Google Cloud Functions [100] and derive the cost the SP has to pay to the cloud provider per MB. Furthermore, based on Amazon Lambda@Edge [13], we derived that data transmission to edge resources (i.e., α^{IP}_{g}) costs about half of the data transmission to the cloud. Using the same approach, i.e., consulting public price lists of Amazon AWS and Google Cloud, we also could derive that one second of execution in the cloud costs about 0.000029 cents, while processing on edge resources is about three times more expensive. This results in $\beta^{\mathrm{SP}}_{\mathrm{CP},g} = 0.000029$ and $\beta^{\mathrm{IP}}_{g} = 0.000087$.

Accuracy in predicting service usage and accuracy in predicting the bargaining partner's cost factors characterize incomplete information. We assume that both stakeholders can predict each cost factor of the corresponding bargaining partner, in an interval of $\pm 10\%$ around the true value (see the predicted cost factors from public sources in Table 4.3). The accuracy of the prediction of service usage is assumed to be $\sigma_g^{\mathsf{p,SP}} = 0.5\sigma_g^{\mathsf{cell}}$, $\sigma_g^{\mathsf{d,SP}} = 0.5\sigma_g^{\mathsf{cell}}$ for the SP and $\sigma_g^{\mathsf{p,IP}} = 1.5\sigma_g^{\mathsf{cell}}$, $\sigma_g^{\mathsf{d,IP}} = 1.5\sigma_g^{\mathsf{cell}}$ for the IP, whereas σ_g^{cell} is a cell specific standard deviation given by the dataset. Therefore, the accuracy of service usage prediction varies between different cells.

In the ToL approach, the IP has to assign a cost factor the SP has to pay for processing on cloudlets (i.e., $\beta^{\mathrm{SP}}_{\mathrm{IP},g}$) and the initial placement of a service (i.e., $\phi^{\mathrm{SP}}_{\mathrm{IP},g}$). Data transmission to the cloud is two times more expensive than data transmission to the cloudlet. Therefore, we assign the cost factor $\beta^{\mathrm{SP}}_{\mathrm{IP},g}$ as a random value between β^{IP}_{g} and $4 \cdot \beta^{\mathrm{IP}}_{g}$, which corresponds to twice the price on the average. We use the same approach to predict a value for $\phi^{\mathrm{SP}}_{\mathrm{IP},g}$ as $unif[\phi^{\mathrm{IP}}_{g}, 4 \cdot \phi^{\mathrm{IP}}_{g}]$, where ϕ^{IP}_{g} is defined as $128\,\mathrm{MB} \cdot \alpha^{\mathrm{SP}}_{\mathrm{CP},g}$ because $128\,\mathrm{MB}$ is the smallest amount that can be used on AWS Lambda@Edge or Google Cloud Functions. Finally, maintenance cost is usually considered amounting to about 10% of the deployment cost [100, 143], which we also use as a reference in our evaluation.

Results

To evaluate the performance of our approach, we analyze the cost reduction, number of cloudlets used, and percentage of game actions processed on cloudlets. Since the cost reductions can vary in size between both stakeholders, i.e., distributed unequally between the two stakeholders, one stakeholder could benefit more than the other. Therefore, we also evaluate and compare the fairness of the approaches in a final step.

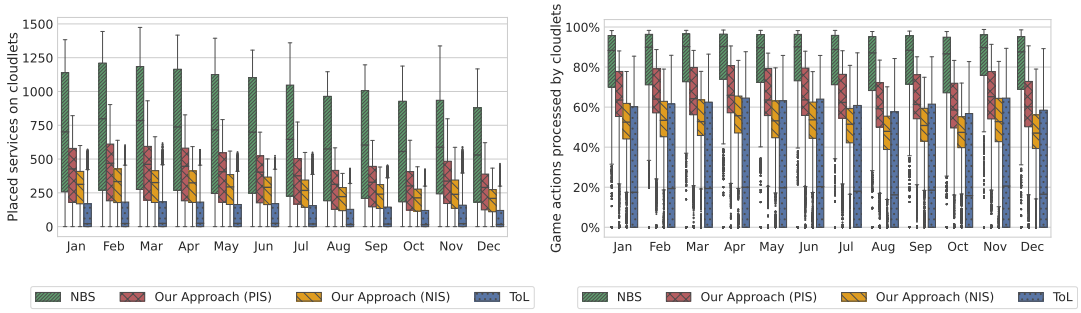
Since placing a service on a cloudlet incurs a fix cost ϕ_g^{IP} for the IP and $\phi_{\mathrm{IP},g}^{\mathrm{SP}}$ for the SP, there is a trade-off between the number of service placements and the number of game actions covered within that time period. Therefore, we simulate every single day by calculating every approach's placement decisions and the associated cost, network traffic, etc. This means that the stakeholders bargain every day. Since our iterative bargaining approach with partial information sharing (PIS) requires the SP to predict the service usage in advance, we add a slight variance based on a half daily standard deviation to the data in our data set to simulate inaccuracies in SP's prediction. In the case of iterative bargaining with no sharing of information (NIS), the IP performs the prediction. However, since the IP does not have internal information about the service, the IP can only roughly predict service usage. Therefore, the IP's prediction is not as good as the SP's prediction. To simulate the inaccuracies of the IP's prediction, we add one and a half standard deviation variance to the data to simulate the IP's prediction, which is an increase of one standard deviation compared to the SP's prediction of the PIS variant.

Cost Reduction Figure 4.5 shows the accumulated cost reductions of all approaches for each stakeholder over one year, where each box shows the accumulated cost reductions at the end of each month. The term "accumulated cost reduction" refers to the amount of money that IP and SP are saving (in our case per month) when using cloudlets compared to a purely cloud-based environment. In essence, it means

$$\text{Accumulated cost reduction} = \sum_{g \in G} \left[\tilde{\kappa}_g^{\{\mathrm{SP},\mathrm{IP}\}} \right] - \kappa^{\{\mathrm{SP},\mathrm{IP}\}} \tag{4.33}$$

Starting with the SP's perspective, the results show that the lower and upper quartiles of the ToL approach (blue boxes) are significantly lower compared to both variants of our iterative bargaining approach (red boxes for PIS, yellow boxes for NIS). However, rare cases exist where the ToL approach performs better than our iterative bargaining (e.g., if the service usage predictions for the iterative approach deviate significantly from the real values). On the average, both iterative bargaining variants lead to higher cost reductions. Furthermore, our PIS variant is quite close to the NBS. From the IP's perspective, the results are comparable. Similar to the previous perspective, our NIS and PIS bargaining approach perform better than the ToL approach and get close to the NBS solution. The mean savings are slightly lower for the PIS variant than for the NBS, but the variance of the cost reduction is larger for the NBS. Furthermore, in some rare cases both graphs show that the PIS and NIS approaches can also achieve negative cost reductions. In other words, both approaches may cause additional cost and do not reduce cost. The explanation for this effect is that both approaches perform their bargaining with the predicted service usage. In some rare cases, the predicted service usage may differ from the actual value in a way that the placement becomes unprofitable, causing additional cost. More specifically, if such a case occurred and additional monthly costs were generated, they ranged, on the average, from \$1.96 (NIS) to \$6.01 (PIS) for the SP and from \$2.88 (PIS) to \$17.01 (NIS) for the IP. In considering the average monthly savings of both approaches, the comparatively low cost that could occur in such an unusual case are rather marginal and negligible.

On average, the SP achieves about 12% higher cost reduction with the NIS variant than with the ToL approach, while the IP achieves about 16% higher cost reduction if the IP also chooses the NIS variant over the ToL approach. When considering the mean values of the



(a) The number of placed services on cloudlets for all approaches

(b) Percentage of game actions processed on cloudlets for all approaches

Figure 4.6: Number of placed services on and game actions processed by cloudlets

PIS variant, the SP achieves about 44% higher cost reduction compared to the ToL approach, while the IP reaches about 47% higher cost reduction if the IP also chooses the PIS variant of our iterative bargaining approach over the ToL approach. The differences in cost reduction are statistically significant (unpaired T-test, $p < 0.01$).

The cost reductions of the NBS (green boxes) are clearly higher than the cost reductions of the ToL approach, as shown by the blue boxes. Using the NBS, the IP can reduce the cost by an additional 78% compared to the ToL approach. The SP can reduce its cost up to about 117% compared to the ToL approach. Since both stakeholders share their cost reductions in the NBS, the cost reductions are identical for both. However, comparing the results of iterative bargaining with the results of the NBS solution, our NIS variant achieves 52% of the possible cost reductions for the SP and 65% of the possible cost reductions for the IP. In contrast, our PIS variant achieves 66% of the possible cost reductions for the SP and 83% of the possible cost reductions for the IP.

All in all, our approach performs better than the ToL cost-based approach in terms of cost reduction and comes close to the optimal solution under complete information.

Number of Placed Services Figure 4.6a shows the number of services placed on cloudlets for all four approaches over one year on a daily basis, grouped by month. The NBS represents the best case with an average number of 624.5 placed services on cloudlets, the ToL approach only achieves about 15% of this result with 94.9 placed services. Using our proposed approach in the NIS variant results in an average number of 248.5 placed services, 39.8% of the best case, leading to 2.6 more placed services compared to the ToL approach. The PIS variant results in an average number of 339.6 placed services, which is 54% of the best case, leading to 3.6 times more placed services compared to the ToL approach. The gaps between the NBS solution and PIS or NIS approaches appear to be quite significant. However, it is worth mentioning that the NBS approach has a 100% prediction accuracy, which results in many placements that are on the verge of profitability. However, considering the number of placed services alone is less meaningful, since a high number of placed services does not necessarily lead to improvements, e.g., in case of a bad placement. Therefore, in the following we consider the average number of game actions performed by the placed services.

Game Actions Processed by Cloudlets Figure 4.6b shows the ratio of game actions processed directly by cloudlets to the total number of game actions. The number of game actions processed by cloudlets can be considered as an indicator for better QoE, since users experience a lower latency. With an average of 94.9 placed services, the ToL approach covers an average of 30% of all game actions, i.e., cloudlets compute 30% of all game actions.

While the NBS approach places 6.6 times more services on cloudlets, only 2.5 times more actions are covered (or 76% of all game actions). This is because service usage is not evenly distributed, with a few cells having very high utilization. Still, the majority of them have medium to low numbers of game actions. Therefore, a 6.6 times increase in cloudlets does not necessarily cover 6.6 times more actions. At this point, it is important to note that 76% coverage of game actions of the NBS approach is the optimal solution, where the joint savings are the highest. A coverage of 100% would also be theoretically possible. However, in our case study, it would not make sense economic, since additional placements would have to take place, which would cause additional cost and reduce the achieved savings. Both variants of our approach achieve between 48% (NIS) and 58% (PIS) of the covered game actions, while the ToL approach only achieves about 30%.

The figure also clearly shows that the lower quartiles of the ToL approach are significantly lower and close to zero compared to other approaches. The reason for this is that in the ToL approach, there are cells where a placement is only advantageous for the IP, e.g., if the price of the IP is too high. However, since the SP decides on the placement, it does not happen since the SP has no advantage. Our approach attempts to reduce cost across all stakeholders and achieves significantly better results than the ToL approach in terms of placed cloudlets and processed game actions.

Overall, the results show that cooperation is beneficial for both stakeholders. The IP can reduce the load in its infrastructure to reduce cost. If the IP also shares its cost reductions cooperatively with the SP, the SP deploys more services, leading to further cost reductions for the IP. The SP also benefits from the cooperative behavior by reducing the cost in the cloud. In total, the number of service placements on cloudlets is higher, which leads to more actions covered by cloudlets and a better quality of service, which in turn is beneficial for the customers of the IP and the SP.

However, full cooperation between SP and IP is rather unrealistic, because it requires a full exchange of information between both stakeholders. Our novel iterative bargaining approach requires a minimum level of cooperation (e.g., exchange of predicted service usage) and yields a solution that comes close to a cooperation with complete information, without requiring the stakeholders to disclose their cost functions or further business-relevant internal details.

Fairness Comparison If the IP shares its cost reductions cooperatively with the SP, the SP is able to deploy more services and further reduces the cost for the IP. This reflects a certain fairness from which both stakeholders can benefit. More precisely, the ratio between the savings of the IP and the SP reflects the fairness of the approaches. In the ideal case, both parties can achieve equal savings, resulting in a high fairness. To measure fairness, we use the well-recognized max-min fairness [30]. It is calculated of each approach using the accumulated savings once at the end of the simulation.

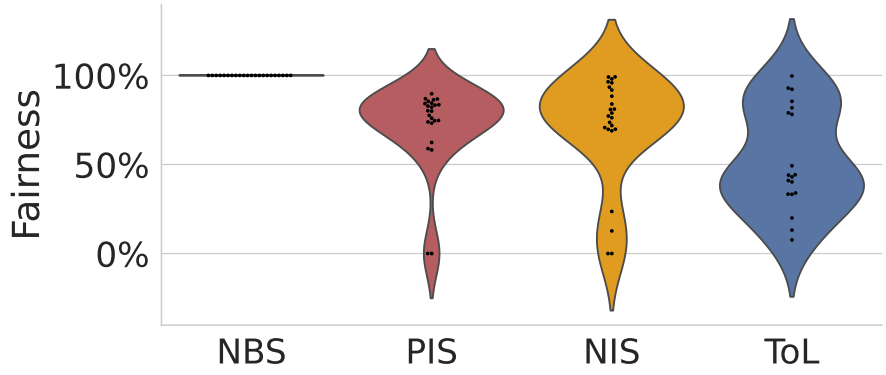


Figure 4.7: Measured fairness of all approaches

Figure 4.7 contains a violin plot showing the distribution of the fairness for each approach. Starting with the NBS, the savings are identical for both stakeholders, resulting in a perfect fairness of 100%. Comparing the fairness of the ToL approach with the fairness of both variants of the presented approach, the fairness of the ToL approach is substantially lower and often below 50%. The reason for this is that in the ToL approach, the IP sets the price, which allows the IP to achieve a relatively high savings, as opposed to the SP. This leads to less fairness between both stakeholders. The distribution of fairness of the NIS and PIS variants are similar. As described in the previous section, the average accumulated savings differ between the two approaches. However, the ratio of accumulated savings between both stakeholders is balanced, which leads to a high fairness of both variants, with some cases, e.g., due to wrong service usage predictions, where the fairness can drop below 50%.

Impact of Incomplete Information The results presented in the previous subsections show that different aspects of incomplete information yield different outcomes in terms of cost savings per stakeholder and fairness of the results. In the evaluation, we showed three different amounts of shared information. Using the NBS, both stakeholders have to share all information they have, whereas using the NIS variant of our approach no information is shared at all. Finally, in our approach using the PIS variant only the SP has to share little information about the service usage, which does not introduce any negative consequences, since the real service usage can be monitored by the IP anyway. These degrees of incomplete information are also reflected in the results. The accumulated cost reductions of SP (Figure 4.5a) and IP (Figure 4.5b) indicate that the more information is shared between the stakeholders, the more cost reductions both can achieve, resulting in NIS giving the least cost reductions and NBS the most, although even the NIS variant achieves better results compared to the traditional ToL approach. This is also visible in the number of placed services (Figure 4.6a) and game actions processed by cloudlets (Figure 4.6b), since these parameters directly influence the cost reductions. The different degrees of information sharing also impact the fairness of the bargaining (Figure 4.7). The NBS, i.e., complete information sharing, always yields an absolutely fair outcome. The less information is shared, however, the more unfair the results will become. In summary, however, the bargaining approach with the lowest level of fairness and with the least cost reduction is still better with respect to all evaluated parameters than the ToL approach, where also no information is shared, but where additionally both stakeholders are not bargaining. This leads to the conclusion that stakeholders should cooperate and share

Table 4.4: Overview of related work

Contributions \ Related Work	[99]	[225]	[50, 180]	[144, 276]	[150, 170, 200, 266]	[123]	[175, 190]	[42]	[184]	[51]	Our Approach
Cloudlet-based edge computing	✓	✓	✓	✓	✓	✓	✓	-	-	-	✓
Considers IP and SP	✓	✓	-	-	-	-	-	✓	-	-	✓
Incomplete information	-	-	✓	✓	-	-	-	✓	✓	-	✓
Cost-awareness	✓	✓	✓	✓	✓	-	-	✓	✓	✓	✓
Evaluation using real data set	✓	-	-	✓	-	✓	-	✓	-	✓	✓

at least a minimum amount of information. This way, they achieve near-optimal results in terms of cost savings, but without having to fully share all information, including private information, which is also reflected in the presented results for the PIS approach. Finally, the PIS approach seems more desirable as it yields better results across all evaluated parameters, but it must be taken into account that in the PIS approach the IP has to trust the SP. The SP could lie about the service usage prediction in its own favor. In scenarios where the SP is not trusted, the NIS variant offers an alternative. Furthermore, it is also possible to extend the PIS variant by a mechanism to enforce truthfulness, such as reputation-based methods [262].

4.2.5 Related Work

Service placement is considered in several publications, where cloudlet-based edge computing takes a large part in recent years, but also other areas like placing services on data caches or edge servers have been investigated.

Table 4.4 presents an overview of related work on service placement and considers aspects of existing approaches compared to our approach, such as whether the papers look at both IP and SP, whether they handle incomplete information, whether they present cost-aware solutions or try to optimize other aspects, and finally whether they use real-world data sets to evaluate their approaches.

Related work in the area of service placement often only focuses on one stakeholder (e.g., only IP or SP), or focuses only on the users. However, it is crucial to understand that stakeholders have different cost and different information about, e.g., the network, available resources or service usage. Therefore, it is important to consider both stakeholders in order to optimize social cost under realistic assumptions. The very few papers that focus on both stakeholders, however, do not handle incomplete information, but rely on complete information sharing between the two players. For example, the work of Gedeon et al. [99] requires all stakeholders not just to know where possible cloudlet locations are, but also what kind of hardware they use, how many resources are available, and how many users are likely to use a particular cloudlet. Similarly, in the work of Shih et al. [225], the SP shares its entire knowledge about its users (e.g., numbers and locations) with the IP so that they can bargain for the price, even if stakeholders in these markets are usually not willing or not able to fully share information.

The work of Cao et al. [42] seems to offer a solution that incorporates multiple stakeholders and incomplete information. However, the authors do not try to optimize the cost for both stakeholders, but only for the IP, while the SP has to either accept the IP's decision or decline it, but they do not interact with each other to find the best solution for both.

The literature on handling incomplete information usually does not take multiple stakeholders into account, but mainly tries to solve problems from one perspective only. For example, the work of Nguyen et al. [180] optimizes the IP's net profit and Zhan et al. [276] try to optimize the IP's operational cost in conjunction with the best achievable user experience. Chen et al. [50] only incorporate collaboration between different IPs. Finally, Ouyang et al. [184] base their placement strategy on information that is only available to the user.

Furthermore, both stakeholders, IP and SP, do not only try to improve their users' service experience, but also try to minimize their operational cost. Thus, it is essential to model the individual cost for both stakeholders and optimize the cost reductions for both stakeholders at the same time. However, most previous works mainly focus on optimizing the users' service experience, like Jia et al. [123], Mukherjee et al. [175] and Peng et al. [190], and do not include the stakeholders' cost at all. Jia et al. [123] offload the workload from the user's device to a cloudlet, Mukherjee et al. [175] reduce power consumption and latency, and Peng et al. [190] maximize the number of served users per cloudlet. Apart from that, the work of Liang et al. [144] tries to optimize the service placement problem by optimizing the placement locations so that the IP's budget is not exceeded.

Finally, large parts of related work evaluate their approaches based on synthetic data and assumptions that often are not based on real-world data sets. For example, Liu et al. [150], Ren et al. [200] and Yang et al. [266] use randomly generated data within ranges that are not motivated by literature or other sources. The work of Mondal et al. [170] models an average Australian city, but not based on a real-world data set, but on hypothetical assumptions. The same also applies to other papers cited above, as shown in Table 4.4. While evaluations based on such data and assumptions are a good starting point to show the general feasibility, evaluations using real-world data sets help to underpin the practicality and applicability of theoretical approaches.

4.2.6 Summary

In this section, a novel iterative bargaining approach between IP and SP for nearly optimal service placement in edge computing scenarios with respect to social cost despite incomplete information was proposed. By being aware of their situation through using publicly available information like public price lists or scientific papers about pricing cloudlets, both stakeholders could project their future situation with respect to their own cost and the cost of the respective other stakeholder and use this projection to decide where to place cloudlets. The proposed bargaining method finds a nearly optimal solution of about 83% of the maximal cost reduction by sharing only service usage predictions and up to 65% of the optimal solution if no information is shared at all. It was also shown how different degrees of incomplete information affect the outcome of the bargaining approach. Finally, the case study based on the mobile AR game Ingress showed that despite incomplete information, this method can achieve up to two times higher game action coverage on cloudlets than a traditional cost model by using a situation-aware approach.

4.3 Predicting the Number of In-game Actions Using Traces of Mobile Augmented Reality Games

Predicting when and where users will play an augmented reality (AR) game can significantly improve deciding on which infrastructure device the service, i.e., the game should be placed, and it is claimed that the situation of the service provider is important for this prediction. In this section, the results of analyzing a dataset of about 23.9 million in-game actions of the mobile AR game Ingress is presented. This dataset of in-game actions is enriched with information describing a service provider's situation with respect to its users (e.g., temperature, precipitation), but also environmental characteristics (e.g., the density of Wi-Fi APs and cellular base stations) during a single in-game action. The results of this analysis emphasize the importance of situation-awareness in the area of edge computing. The results show that up to 84% of the variation in the number of in-game actions can be explained when being aware of the situation. Furthermore, the analysis shows that information that represents time (e.g., weekday), physical conditions (e.g., weather), computational environment (e.g., density of Wi-Fi APs), and user information (e.g., game incentives) significantly influence the number of in-game actions.

The remainder of this section is organized as follows. Subsection 4.3.1 introduces and explains the collected data, which is analyzed in Subsection 4.3.2 concerning available information and the derived situation. Finally, Subsection 4.3.3 reviews related work.

Parts of this section have been published previously [81].

4.3.1 Dataset

In this section, we present our dataset by explaining the used collection methods and the corresponding data sources.

Data Basis

The dataset contains in-game actions of the popular AR game Ingress. The current state of the game and users' in-game actions are visible within the Ingress app, but also on the so-called Ingress Intel Map website[1], which contains the current state of all portals worldwide. To collect this data, we built a crawler based on Python and Selenium that automates browsers and requests changes in the game state every second. Important for us, these changes include user interactions with the portals, which have a clear geographical position. This interaction makes it possible to determine the position of a user since a user needs to be in the immediate proximity ($\leq$ 40 meters) of the portal to interact with the portal.

Regarding the spatial extension, the dataset covers a large part of the federal state of Hesse in Germany with the Rhine-Main area in the center of the observation area. The Rhine-Main area is the third-largest metropolitan area in Germany, with a total population exceeding 5.8 million. Within this area, the three largest cities are Frankfurt (about 670,000 citizens), Mainz (about 209,000 citizens), and Darmstadt (about 142,000 citizens). The perimeter of

[1] https://www.ingress.com/intel

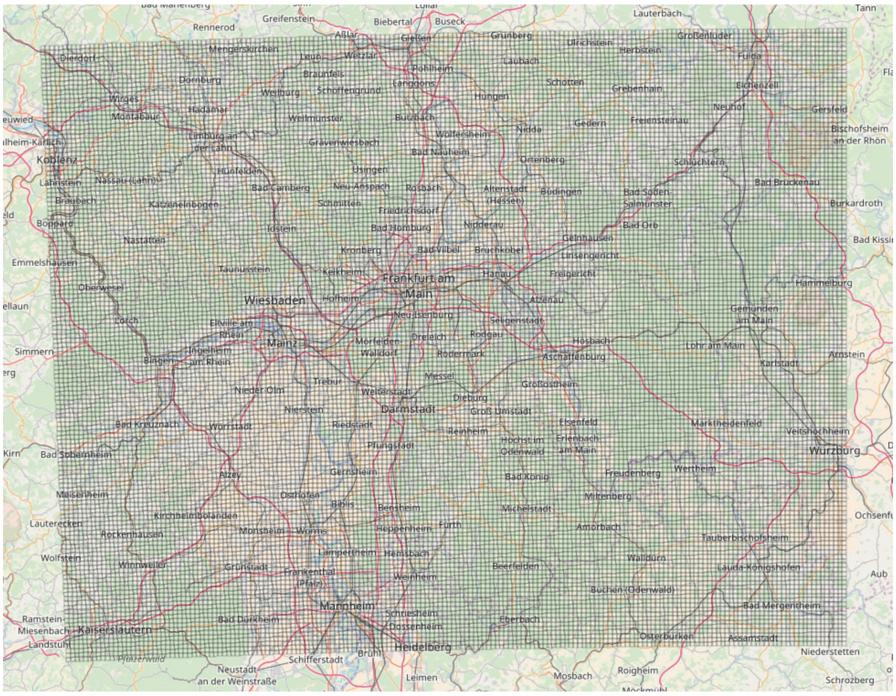


Figure 4.8: Observation area

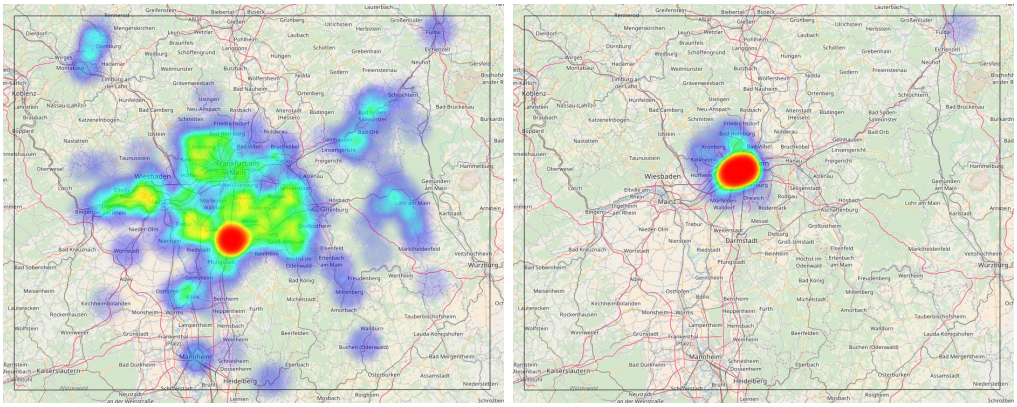
the observation area is 140 km on the latitude and 180 km on the longitude, resulting in a total area of about 25,200 km^2. We use a grid size of 1 km $\times$ 1 km to analyze the data with a fine granularity without infringing on the individual user's privacy. Figure 4.8 visualizes the observation area and the division into the grid.

Within the observation area, we have recorded the in-game actions as interactions of the Ingress users with the respective portals. Each portal has a geographical assignment as a WSG84 coordinate. Consequently, the users' positions are recorded implicitly by the interaction with the portals. To protect the privacy of individual users, we do not provide the exact location of the portals where the interaction took place, but the grid in which the portals are located. Therefore, the motion profiles have a resolution of 1 km $\times$ 1 km within an area of 25,200 km^2. Figure 4.9a and Figure 4.9b illustrate the motion profiles for two users with different mobility behaviors in the form of a heat map. Furthermore, the figures also illustrate that despite the 1 km^2 grid, the mobility traces are still quite accurate, which can be used to classify users or places of users.

We provide the data for a period of almost 1.5 years, starting on the 1st of January 2016 until the 31st of May 2017. For this period, in-game actions of Ingress are complete, except for a maintenance phase from September 20th until September 25th, 2016.

Contextual Information

We enriched the Ingress data by adding data for various contextual information. This combination of data can help us gain a deeper understanding of the relationship between situation and in-game actions. We classify this information into static and dynamic factors as well as macro-



(a) High mobility **(b)** Low mobility

Figure 4.9: Users with high activity and different mobility

and micro-level factors. Static factors are not subject to changes over time; they are constant for the period under consideration (e.g., the number of cellular base stations). Dynamic factors are subject to change and are present in a temporal resolution (e.g., the temperature on an hourly basis). Macro- and micro-level factors describe the spatial resolution of the factors. Here, macro-level factors refer to factors that relate to the entire area of observation, can change over time, but do not differ significantly at the grid level. For example, public holidays occur over time, but the factor is identical for the entire area of observation. In contrast, micro-level factors refer to the individual 1 km $\times$ 1 km grids of the observation space. For example, the amount of precipitation is available in tenths of mm for each grid. In addition to the spatial and temporal properties of the information, we also consider the four dimensions of contextual information by Bellavista et al. [25] and assign our information to these dimensions. These four dimensions consist of the *physical*, the *computing*, the *time*, and *user* information, which will be explained below.

Physical Information Physical information describes the physical conditions, e.g., weather. With regard to physical information, we collected high-resolution weather radar data from the German Meteorological Office (Deutscher Wetterdienst, DWD). The data provides information about the amount of precipitation (in tenths of mm) on an hourly basis and has the same resolution and position as the grid. Air temperature and humidity are also available and come from a weather station provided by the DWD. This weather station is located in Frankfurt and thus in the center of the observation area and represents a macro-level situation. It is noteworthy that temperature and humidity are only available as macro-level information. At this point, we expect no major deviations in temperature and humidity between the grids, in contrast to precipitation, which often has a local influence. However, precipitation can be aggregated to a macro-level information by calculating the sum or the average precipitation of all grids on the micro level. To describe the environment at the micro level, we also have enriched data on land use and urban characteristics, such as the number of public facilities that may lead to differences in in-game actions. We also looked at public facilities such as universities and schools, as well as places with attractions and entertainment establishments. Here, we used data from OpenStreetMap (OSM) to enrich the dataset with information on Points of Interests (POIs) that describe places that may influence the game behavior (e.g.,

Table 4.5: Data sources

Name	Source
Ingress Activity	`https://www.ingress.com/intel`
Weather	`https://www.dwd.de`
Precipitation	`https://www.dwd.de`
OSM POIs	`https://www.openstreetmap.org`
Wi-Fi APs	`https://wigle.net`
Base Stations	`https://www.bundesnetzagentur.de`

train stations, universities, public parks, playgrounds). Furthermore, we provide a script in addition to the dataset, which allows adding information on other types of POIs. This can help the scientific community to adapt the dataset for specific research questions if necessary.

Computing Information Computing information captures the technical facilities of the environment, such as the density of Wi-Fi Access Points (APs). Our dataset includes information on the density of Wi-Fi APs per grid collected from www.wigle.net, which is one of the largest Wi-Fi war-driving communities worldwide. Furthermore, we include the number of cellular network base stations per grid collected from the Bundesnetzagentur (BNetzA), the German federal network agency.

Time Information Time information incorporates various time-related factors, e.g., time of the day or public holidays. The dataset includes information on the time of the day, the day of the week, and seasonal effects. In addition, the dataset also provides information about local public holidays.

User information User information denotes high-level aspects related to the social dimension of users, such as user preferences, people nearby, and the current social environment [4]. Concerning the user, we included information on special Ingress events like so-called "mission days". These events are often organized together with the local tourism association to explore a city and the sights. These Ingress events often attract thousands of users from different places who then move through a city together. In addition to mission days, we provide information about so-called anomalies in the dataset. These anomalies represent rule changes in the game, such as double points, and are likely to affect the players' behavior.

Table 4.5 provides a brief summary of the data sources in the dataset.

4.3.2 Analysis

Our analysis aims to investigate the relationships between situations and in-game actions. The analysis consists of three parts, beginning with an overview and description of the main variables, followed by macro- and micro-level regression analysis examining the relationships between situations and in-game actions. The macro-level analysis aims to predict in-game actions for the entire area of observation, whereas the micro-level analysis examines the influence of situations at the grid level.

Table 4.6: Descriptive statistics of the dataset

Variable	Mean	SD	Minimum	Maximum	Total
Users				24,316	
Actions				23,903,293	
Actions/Day	46,686.12	20,690.24	14,926	409,793	
Actions/User	983.03	4,834.11	1	13,9567	
Portals/Grid	2.11	7.99	0	215	53,259
Access Points/Grid	103.82	694.13	0	27,899	2,616,213
Base Stations/Grid	0.25	0.84	0	31	6,174
Temperature	10.16	7.83	-12.6	34.9	
Portals/Day/Grid	15.84	38.33	0	1,229	

Overview

Starting with the summary statistics, Table 4.6 shows an overview the main variables. During our data collection period, we gathered more than 23.9 million in-game actions from over 24,000 users. Within our observation area of 25,200 km^2, there are over 53,000 ingress portals the users of the game can interact with. On the average, each grid has about 100 Wi-Fi APs, and every fourth grid has a cellular base station. In the collection period, the average temperature was 10.2°C, and the average daily precipitation was 1.56 mm/h.

Macro-Level Analysis

Our macro-level analysis primarily considers contextual information that affects the entire area of observation. We estimate Equation (4.34) using a least squares regression with robust standard errors to explain the number of in-game actions, $ing\_actions$. Our model explains the in-game actions based on factors of each dimension of information: $Physical$ information such as precipitation, temperature, temperature^2 [2], and humidity, $Time$ information such as day of the week and public holidays, and $User$ information such as game-specific events.

$$\begin{aligned} ing\_actions_t = \alpha_0 + & \sum_{p \in P} \beta_p \cdot Physica_{pt} \\ & + \sum_{k \in K} \gamma_k \cdot Time_{kt} \\ & + \sum_{u \in U} \delta_u \cdot User_{ut} + \epsilon \end{aligned} \tag{4.34}$$

We extend the model step-wise by adding additional information dimensions. This allows us to compare the information dimensions with regard to their explanatory power. In the first step, the model contains only the constant (α) and thus simply estimates the average number of in-game actions. This model constitutes the baseline for a comparison of the following models that additionally contain information of different dimensions.

[2] We assume that the effect of temperature follows a quadratic function rather than a linear function.

Table 4.7: Macro level regression results of Models 1 and 2

	Model 1		Model 2	
Variable	Coef.	SD	Coef.	SD
Trend			-24.10**	8.53
Weekday =				
Monday			-1,567.46	5,276.39
Tuesday			-2,183.13	5,349.45
Wednesday			-2,106.73	5,383.16
Thursday			-850.63	5,330.91
Friday			-2,340.52	5,281.10
Saturday			-753.39	5,786.99
Pub. Holiday			-7,285.36*	3,367.48
Intercept	46,686.12***	915.28	54,539.47***	3,561.24
Num. of obs.		512		512
Prob >F		0.000		0.000
adj. R^2		0.000		0.0357
Root MSE		20,710		20,498
*: $p < 0.10$, **: $p < 0.05$, ***: $p < 0.01$				

Tables 4.7 and 4.8 presents the estimation results of all individual models. The results of the F-test ($p < 0.001$) for all four models show that at least one of the variables can significantly explain some variation in our dependent variable.

The root mean square error (RMSE) of the first model measuring differences between estimated values by a model and the values observed indicates a high deviation. In line with this observation, the adjusted R^2 amounts to <0.00% and indicates that the model explains the variance of the in-game actions only poorly.

The second model additionally considers factors of the $Time$ dimension. The results show that these factors lead to a slight reduction of the RSME, in addition, the adjusted R^2 shows that this model captures 3.57% of the variance of the in-game actions. Furthermore, the results show a negative trend ($p < 0.05$) in in-game actions and a negative impact of public holidays ($p < 0.10$). This shows that the extension by additional contextual information of the $Time$ dimension leads to a slight improvement of the estimation quality.

The third model also includes factors of the $Physical$ dimension. Looking at the adjusted R^2 of the third model in comparison to the second model, it becomes apparent that the consideration of the $Physical$ dimension explains additionally up to 9.31% of the variance and significantly improves the explanatory power of the model. On the one hand, this indicates the high explanatory power of the $Physical$ dimension. On the other hand, it shows that the use of mobile AR games strongly depends on physical information. Thus, temperature ($p < 0.05$) and precipitation ($p < 0.1$) have a significant impact on the number of game actions and thus on in-game actions.

Finally, the fourth model contains factors of the $User$ dimension. Here, we essentially consider factors such as game-specific events like mission days and anomalies. The latter are changes

Table 4.8: Macro level regression results of Models 3 and 4

	Model 3		Model 4	
Variable	Coef.	SD	Coef.	SD
Trend	-27.12***	8.01	-35.95***	3.30
Weekday =				
Monday	-1,487.45	5,133.27	3,481.57*	1,377.69
Tuesday	-1,527.50	5,105.70	3,058.44*	1,373.02
Wednesday	-2,366.97	5,238.52	2,477.78	1,402.93
Thursday	-1,395.95	5,316.15	3,793.64**	1,346.51
Friday	-1,984.58	5,042.42	2,635.81*	1,325.45
Saturday	-94.61	5,542.18	2,904.54	1,646.90
Pub. Holiday	-8,794.87***	2,666.87	-6,609.91**	2,305.84
Temperature	1,538.74***	207.69	1,189.14***	126.77
Temperature^2	-74.14***	12.55	-51.81***	5.33
Humidity	-355.68*	168.13	-158.66***	32.86
Precipitation	-41.08	21.43	-59.10***	12.91
Ing E.			367,917.80***	1,698.71
Ing. E. (L+1)			143,189.60***	1,923.38
Ing. E. (L+2)			15,443.17***	1,443.23
Ing. An. 1			22,340.81***	1,803.41
Ing. An. 2			3,611.02***	748.15
Intercept	78,515.16***	15,708.01	60,110.68***	2,868.27
Num. of obs.		512		512
Prob >F		0.000		0.000
adj. R^2		0.1288		0.8446
Root MSE		19,562		8,313
*: $p < 0.10$, **: $p < 0.05$, ***: $p < 0.01$				

in the rules of the game, such as double points for actions (anomaly 1) or "no energy"[3] (anomaly 2). With regard to the latter, Niantic shortened the supply of energy which made it on the one hand more difficult to defend the portals, but on the other hand, it also resulted in fewer resources to attack the enemy portals. Overall, this led to lower activity in the game. Such events and rule changes often come with additional incentives (e.g., virtual badges), which can affect user behavior. The results for the fourth model show a significant increase in the explained variance and a reduction of the RSME by 57.5%. Also, the results show that the high explanatory power of $User$ dimension not only explains a large part of the variance in the in-game actions but also that additional factors of the $Time$ dimension become significant since we now control for strong outliers caused by the specific $User$ information.

To develop an understanding of the individual factors and their relationship to in-game actions, we examined the individual contextual information of the last model in more detail. The first group of variables, $Time$, measures the relation between temporal factors and in-game actions. Our results show a significant negative trend of the in-game actions, which suggests that gaming activity generally decreases over time. Concerning weekday variables, we see some significant weekday effect, since the in-game actions for Monday ($p < 0.1$), Tuesday

[3]To perform an in-game action, a particular amount so-called *energy* or *exotic matter* is required.

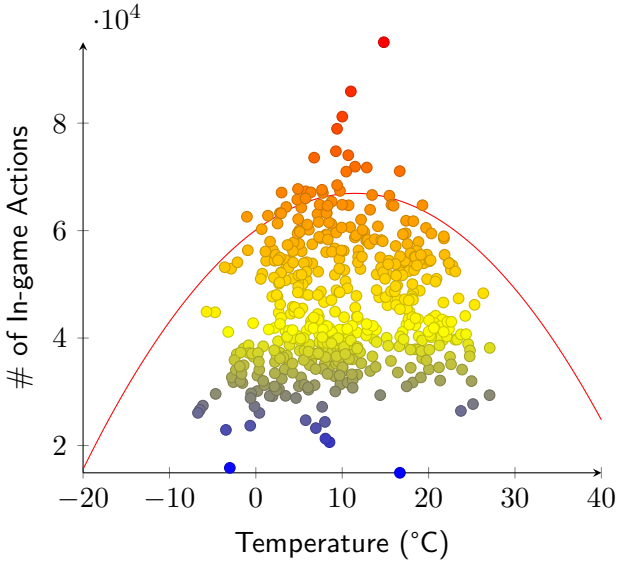


Figure 4.10: Relation between temperature and in-game actions

($p < 0.1$), Thursday ($p < 0.01$) and Friday ($p < 0.1$) are significantly higher in comparison to the omitted weekday Sunday. In-game actions on the individual weekdays show that the average number of in-game actions on all days is similar, but that the deviation increases on weekends, especially on Sundays. Our findings also reveal a negative relationship between public holidays ($p < 0.01$) and in-game actions.

The second group of variables, $Physical$, measures the relation between weather factors and in-game actions. Our results show that precipitation has no significant relation to the in-game actions. Furthermore, temperature ($p < 0.01$) is positively related to in-game actions. As expected, temperature^2 ($p < 0.1$) is negatively related to in-game actions, which suggests a U-shaped nonlinear relationship between temperature and in-game actions. We have visualized the estimated coefficients for temperature and temperature^2 in Figure 4.10. This figure shows that there is an ideal temperature range around 10.5°C at which activity is at its highest level. As soon as the temperature rises or falls, the number of actions decreases, meaning that if the outside temperature is too hot or too cold, in-game actions are reduced because fewer users go outside to play Ingress. Further, we also find that humidity ($p < 0.01$) is negatively related to in-game actions. In contrast to temperature, humidity is rather linear and the number of in-game actions decreases with increasing air humidity (see Figure 4.11).

The results of the third group of variables, $User$, shows a significant impact of all measured variables. Especially game events are highly significant ($p < 0.01$) and positively related to in-game actions. During game events, the number of in-game actions increases more than seven times compared to the average number of in-game actions. Since we know that such events attract users from the surrounding cities and even foreign countries, we have added two lead variables that capture the variance of the in-game actions on the previous days. Here, we suspect that users will arrive a few days in advance due to their long journey and this will also affect in-game actions. Our results support the assumption and also show a significant increase ($p < 0.01$) on both preceding days. Also, regarding rule changes in the game (or Ingress anomalies), the results show logically justifiable values. Anomaly 1 (double points for

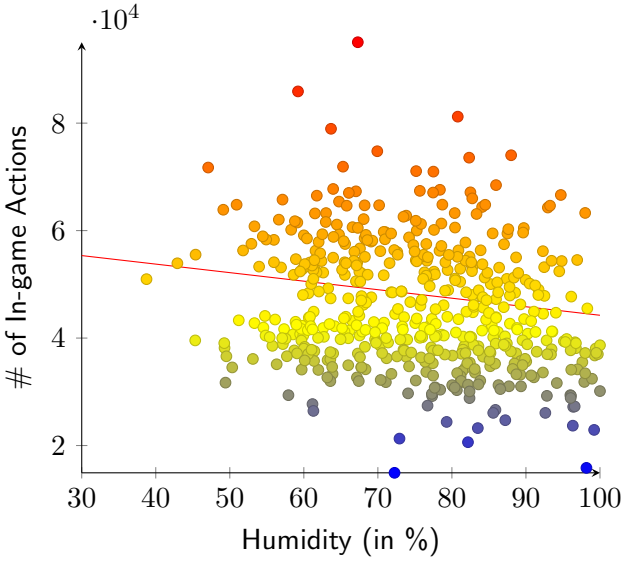


Figure 4.11: Relation between humidity and in-game actions

all game actions) shows a significant ($p < 0.01$) positive relation to in-game actions, while anomaly 2 (no energy supply) shows a significant ($p < 0.01$) negative relation. The results support our hypothesis that these game-specific events affect user preferences, while double points provide a clear incentive and are in a positive relationship with in-game actions, the unavailability of energy supply decreases in-game actions because the users may avoid game actions that drain their energy points.

Our results indicate a strong relationship between contextual information and the number of in-game actions. These findings suggest a substantial influence of situations on location-based services, such as mobile AR games, which are predominantly used outdoors. Furthermore, the results indicate a significant influence of the user's situation and properties that are related to the user's preferences and in-game actions. From the point of view of the game provider, these properties do not represent exogenous factors, as do most other contextual information (e.g., temperature, humidity). In fact, these factors (such as rule changes and events) are planned and operated by the game provider and are therefore predictable.

To test the predictive power of situations, we forecast the number of in-game actions for one month with a forecast of one day. For this purpose, we also use a regression model and apply an out-of-sample prediction. The model is estimated based on data from the first 12 months (January 2016 to December 2016) and predicts the number of in-game actions for the following month (January 2017).

Figure 4.12 illustrates the observed number of in-game actions (red line) and the results of the out of sample prediction for models 2, 3, and 4. The graph shows how the prediction improves with an increasing number of information dimensions at the observed value. Starting with the $Time$ dimension, there is a recurring weekly pattern with a negative trend (RSME: 16,284.36). The $Physical$ dimension improves the prediction even further (RSME: 8,364.58) and emphasizes the extremes. The $User$ dimension also leads to an additional improvement and reduces the root-mean-square error. The calculated RSME for the prediction period amounts to 3,222.31 and is even lower than the RSME of the previously estimated models.

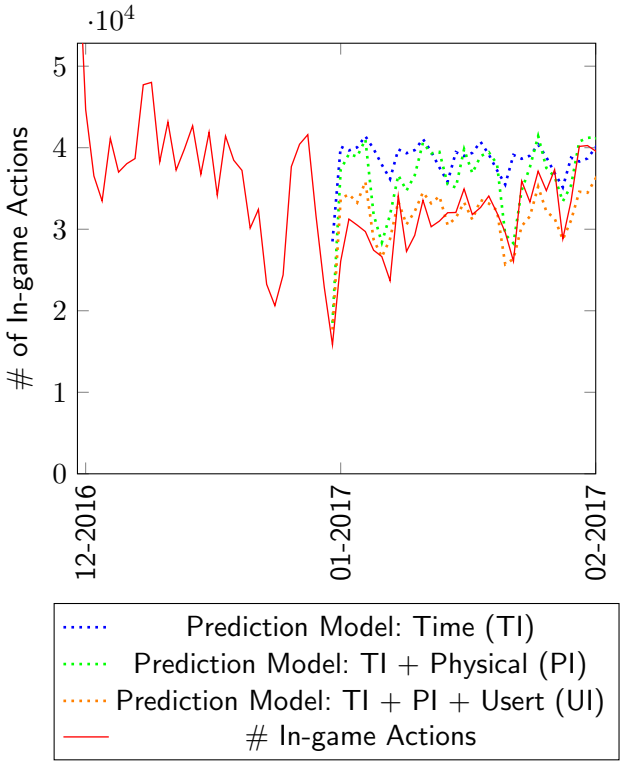


Figure 4.12: Prediction of the number of in-game actions

The reason for this is probably hidden because there were no in-game events or similar events during this period that would otherwise cause a high variance.

In summary, our macro-level analysis indicates a strong relationship between the situation and the number of in-game actions. Furthermore, we have demonstrated the predictive power of situations in terms of the number of in-game actions. In order to analyze further influencing factors that have a local influence, we analyze contextual information at the micro level in the following subsection.

Micro-Level Analysis

In the third part of our analysis, we examine the relationship between situation and the number of in-game actions at the micro level. We use negative binomial regression to examine this relationship. Negative binomial regression is a common method for modeling count variables, in particular for over-dispersed count outcome variables.

Our regression model essentially consists of three variable groups: $Physical$, $Time$, and $Computing$ (see Equation (4.35)). These variable groups contain the variables of the dataset presented in the previous section, where i is the grid indicator and t the day during the observation period.

Table 4.9: Regression results (Micro level)

Variable	IRR	Mean
Trend	0.9992***	0.0000
Weekday =		
Monday	1.1690***	0.0048
Tuesday	1.1196***	0.0046
Wednesday	1.1151***	0.0046
Thursday	1.1045***	0.0046
Friday	1.1400***	0.0047
Saturday	1.0521***	0.0043
Public Holiday	0.8217***	0.0057
Temperature	1.0102***	0.0005
Temperature^2	0.9995***	0.0000
Humidity	0.9973***	0.0001
Precipitation	0.9998***	0.0000
Attractions	1.0245***	0.0012
Wi-Fi APs	1.0001***	0.0000
CBS	1.1029***	0.0009
Ingr. Event	212.5932***	3.4944
Ingr. Anomaly 1	1.0685***	0.0073
Ingr. Anomaly 2	1.0507***	0.0034
Intercept	0.1391***	0.0013
N		13,028,400
Groups Grids		25,200
*: $p < 0.10$, **: $p < 0.05$, ***: $p < 0.01$		

$$
\begin{aligned}
ing\_actions_{ti} = \alpha_0 &+ \sum_{p \in P} \beta_p \cdot Physical_{pti} \\
&+ \sum_{k \in K} \gamma_k \cdot Time_{kti} \\
&+ \sum_{u \in U} \delta_u \cdot User_{uti} \\
&+ \sum_{c \in C} \zeta_c \cdot Computing_{cti} \\
&+ \epsilon_{ti}
\end{aligned} \tag{4.35}
$$

Table 4.9 presents the results of the model estimated on in-game actions on the micro level reported as an incidence-rate ratio (IRR).

Starting with variables related to $Physical$ information on the micro level, we also find significant relationships for all variables used here. Due to multicollinearity, we have deliberately dropped variables for the following analysis. For example, the number of Wi-Fi APs of the $Computing$ dimension provides a good proxy for population density within a grid, while

places with a high population density also have a high density of restaurants, cafés, and bars that are part of the $Physical$ dimension.

By considering weather-related variables, we find a significant influence also on the micro level. In particular, precipitation and humidity ($p < 0.01$) shows a significant influence. According to our estimations, a one unit increase in precipitation (measured in tenth of mm) decreases the in-game actions on average by a factor of 0.9995. Also, areas with tourist attractions ($p < 0.01$), which usually increase the attractiveness of a place, have a positive influence on the in-game actions at this place. A point of interest increases the in-game actions on average by $+2.56\%$ for the given grid.

Followed by the factors of $Time$ information, the results reveal a statistically significant ($p < 0.01$) weak negative trend in the in-game actions. This finding is also consistent in the analysis of macro-level. With regard to weekday effects, the results show a higher activity for all weekdays compared to Sunday. The effect seems to be greatest for Mondays and Fridays. Similar to the macro-level analysis, we also see a negative relation between in-game actions and public holidays at the micro level. According to our estimations, public holidays decrease the in-game actions by 16.7%.

With respect to the $Computing$ dimension, grids with many cellular base stations ($p < 0.01$) and Wi-Fi APs ($p < 0.01$) show significantly higher numbers of in-game actions. The estimates show that the effect for cellular base stations is significantly higher than for Wi-Fi APs. This is probably due to the fact that the mobile network is the only way to access the Internet, whereas the density of Wi-Fi access points has only a weak effect.

Regarding the $User$ dimension, the results show a significant influence of local in-game events on the in-game actions within a grid. If an event takes place within a grid, the number of in-game actions increases by a factor of 212.6 ($p < 0.01$). Furthermore, with regard to anomalies, the results indicate a significant ($p < 0.01$) positive effect of both types.

In summary, a large part of the contextual information of both analyses shows a highly significant relation. Beyond that, the models show a high explanatory power of situations with regard to in-game actions at both the micro and macro level. This emphasizes the importance of situation-awareness, especially in augmented reality applications, which, compared to other applications, often take place outdoors and are particularly situation-dependent. Furthermore, the results of our predictions show how the number of in-game actions can be forecasted accurately based on the situation on the micro and macro level.

4.3.3 Related Work

Service Usage Prediction and Context

Predicting the number of in-game actions can be considered a special case of predicting service usage, which has a long tradition in computer science and is still an area of recent research. Researchers have proposed various approaches to predict, for example, CPU and network load [70], both in the context of grid computing [277] and in cloud computing infrastructures [41, 121, 169], as well as the utilization of services [229] and infrastructures such as computer networks [142] or power networks [125].

For our work, predicting the load in cloud computing infrastructures [41, 121, 169] is of particular interest. In general, the approaches proposed in the literature use the ability of dynamic resource allocation in cloud infrastructures in order to optimize resource allocation and utilization [41, 169] and to reduce power consumption of the cloud computing hosts [27, 103]. Accurate predictions are necessary to reallocate resources of services according to their upcoming service usage [41, 121, 169]. However, the majority of approaches use internal data as sources of information for prediction, i.e., information such as the used resources (e.g., CPU utilization, memory, and storage) in combination with temporal properties to predict future utilization. In general, these approaches attempt to predict future service load by historical load patterns and trends over time. However, this neglects context factors that also represent an external source of information and can have an impact on service users and service usage.

In recent years, context-awareness for mobile devices and applications has gained attention due to the increased availability of context factors measurable by mobile sensors (e.g., GPS, temperature, acceleration). Researchers use contextual factors to optimize predictions in various areas, for example, to predict the usage of mobile applications based on context factors [226], provide recommendations based on a context-aware rating prediction [201], or predict the most probable activity using context factors [202]. However, approaches in this area usually consider individual users and are thus influenced by the preferences and usage patterns of individual users. Hence, it is difficult to predict the general service usage or load from individual usage patterns only.

Only a few approaches consider the effect of context factors for the prediction of infrastructure usage or load on the overall level. In particular, approaches to predict the power consumption or load [125] often use context factors such as weather conditions to improve their prediction accuracy. These approaches are based on the assumption that users, for example, use other electronic devices depending on the weather, which explains the variance in power consumption. Furthermore, researchers have shown how to use user roles for context-aware service recomendations [259]. Paasovara et al. [186] show that mobile AR games also incorporate user interactions outside the game, which is part of the social context and can influence the usage of mobile AR games, e.g., during in-game idle times. In other areas, Hinz et al. [110] show that the context of TV consumption can influence economic behavior in online auctions. All these studies demonstrate the influence and the predictive power of context factors.

Overall, the use of context factors in the prediction of service usage is not very common, although the optimization potential and application areas are manifold. Our study addresses this gap in the literature by contributing in two ways. First, through our data analysis, we highlight the role of context factors in the presented scenario, which serves as an entry point for further research. Second, we provide a valuable dataset containing the service usage of a popular AR game with mobility patterns and additional context factors. This dataset can be used for context-related research, such as context-aware placement of caches and cloudlets.

Dataset Application Areas

Predicting service usage is useful for several applications and use cases that can be evaluated using our dataset, e.g., cloudlets. Cloudlets bring storage and computing capabilities closer to

the users. As a consequence, data and computations can be offloaded [210]. Thus, cloudlets support low-latency services, and existing communication infrastructures such as wireless access points [166, 210], radio access networks [1], or femtocells [47] can be used to deploy cloudlets. Liu et al. [151] show that cloudlet placement, as well as offloading data to mobile edge nodes, is a promising strategy to improve network efficiency.

Taking into account context factors is important for this area of research. One stream in the literature focuses on the question of where to ideally place cloudlets under the constraints of capacity utilization and costs for acquisition [37, 38]. Yang et al. [263] discuss the basic service placement problem as well as the cost-aware service placement problem of cloudlets as an instance of considering the computing context.

Researchers often use existing datasets or combine different datasets to evaluate these approaches. For example, Yang et al. [263] use an existing dataset of taxicab movements and combine this dataset with a demand distribution for certain services. Both data sets together simulate the demand for certain services at certain locations as a user moves through a city. The dataset of GPS trajectories of taxicabs published by Yuan et al. [270] is a widely used dataset for different evaluation purposes in the fields of data and knowledge engineering, machine learning, mobile intelligence, navigation, and transport [173, 271, 279, 280].

However, this combination of datasets (i.e., mobility traces with demand distribution) does not consider the dependency of the context. This means that the distribution of the service demand is identical for all locations, and the overall demand increases depending on user traces in a location. However, this combination of datasets ignores the fact that the demand for certain services is also affected by context factors such as location. For example, the service demand for a mobile app containing places of interest in the state of New York might be higher in New York City than in surrounding cities. Therefore, the distribution of the demand for services also depends on the location and is not identical for all locations. However, there are numerous other factors such as weather, daytime, etc. that can have an impact not only on service demand but also on the mobility behavior of users. All these factors are context factors and provide information about the situation in which a user uses a certain service or not. Our dataset addresses this problem by providing a variety of measured context factors with respect to the time and location of service usage. This supports the evaluation of various context-aware applications and use cases, such as context-aware cloudlet placement strategies.

4.3.4 Summary

The case study based on Ingress highlighted metrics and measures for situation-aware mobile AR applications. In the course of the analysis, the relationship between the service providers situation and the number of service usages was investigated. The results show that more than 84% of the variation of the number of in-game actions can be explained by being aware of the situation. This shows the enormous influence of the situation on the number of in-game actions. Based on the situation, the number of in-game actions for one month was predicted with high accuracy.

4.4 Summary

This chapter contributed to the novel concept of situation-aware infrastructure edge computing. It was shown how a service provider can leverage the concept of situation-awareness to decide on which infrastructure devices a service should be placed by perceiving information about its own situation regarding providers of infrastructure devices and the users.

In particular, the following contributions were presented:

- *Multi-Stakeholder Bargaining:* A situation-aware approach to support economic decision about on which infrastructure devices to place a service despite incomplete information.
- *Predicting In-Game Actions:* Using information regarding the users of a service to derive the service provider's situation for deciding where a service should be placed.

5

Situation-aware Device Edge Computing

This chapter presents two approaches using situation-awareness in the area of device edge computing. Section 5.1 motivates the use of situation-aware device edge computing. Section 5.2 introduces an approach where mobile devices in opportunistic networks use their situations to decide about which mobile devices in the network a process should be offloaded to. After that, an approach is presented in Section 5.3 that projects a mobile device's future situation regarding its connection state. It is predicted whether a device's Wi-Fi connection is lost within a given time frame and decided whether a proactive handover to a cellular connection should be performed or not. Section 5.4 concludes this chapter.

Parts of this chapter have been published previously [112, 233].

5.1 Motivation

One of the features of device edge computing is that processes are offloaded from mobile devices to other mobile devices in the same network. This provides mobile devices a collaborative computation and storage resource pool in their proximity. Here, the main goals are to reduce latencies, to provide computation resources when cloud- or infrastructure devices are either not available or not feasible, and to reduce unnecessary data transfers through network backends to clouds if the computation can be done in proximity of the offloading mobile device [269]. There are many use cases for device edge computing, e.g., face detection in emergency scenarios could help rescue helpers to save resources for essential communication [62], and environmental monitoring with mobile sensor nodes is a current research topic [24]. Situation-aware decisions with respect to the question on which mobile devices in the proximity a process should be offloaded to can lead to significantly reduced latencies and execution times and a more fair distribution of multiple processes within the network of mobile devices. Further, situation-awareness in device edge computing improves the selection of mobile devices, since compute nodes that are especially suitable to execute a given process improve the completion rates of offloaded processes. Finally, connections between mobile devices, but also between a mobile device and for example a Wi-Fi access point, can be interrupted due to the mobility of the devices. With respect to this aspect, a situation-aware approach can reliably predict such connection losses so that decisions regarding countermeasures like setting up alternative connections in advance can be made confidently.

Figure 5.1 shows how both above presented problems, i.e. (i) finding the best available mobile devices to offload a process on and (ii) predicting connection losses in advance, are solved using the novel concept of situation-aware device edge computing. To cope with

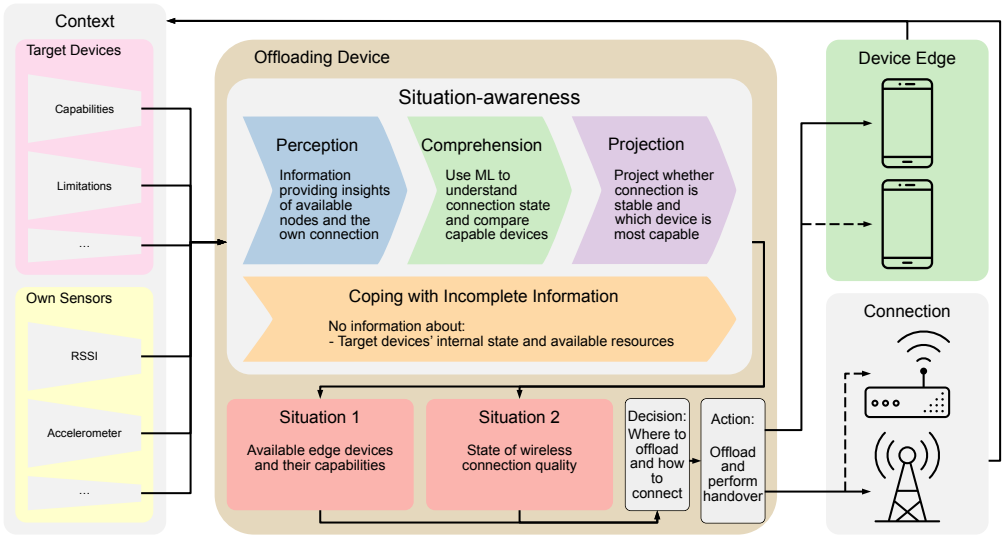


Figure 5.1: Situation-aware device edge computing

problem (i), a mobile device perceives information published by available mobile devices within the network and comprehends its own situation with respect to which devices are available to execute processes on and their capabilities and limitations. This method of mobile devices announcing their capabilities and limitations copes with two challenges in device edge computing applications. First, due to the heterogeneity of mobile devices with respect to operating systems, CPU architectures and available hardware components, announcing this information aids in using only capable devices. Second, this information is not perceivable from the outside, leading to incomplete information. By letting the mobile devices publish this information, offloading devices can cope with this incomplete information. Based on comprehended current situation the offloading device's future situation is projected with respect to which mobile device is able to execute the task and likely is able to return the result in a given time. Finally, it is decided on which device a process is offloaded to, indicated by the actions taken in Figure 5.1, where the dashed arrow is the option that was not chosen. For (ii), a mobile device perceives information that describe its situation with respect to the own connection status like the signal strength of the wireless connection but also velocity and altitude. Based on this information, the current situation of the connection is comprehended and the future situation with respect to the connection state is projected. The projection is used to decide whether an alternative connection should be established in advance. In Figure 5.1, the device decides to establish a connection to the base station and leaving the Wi-Fi access point.

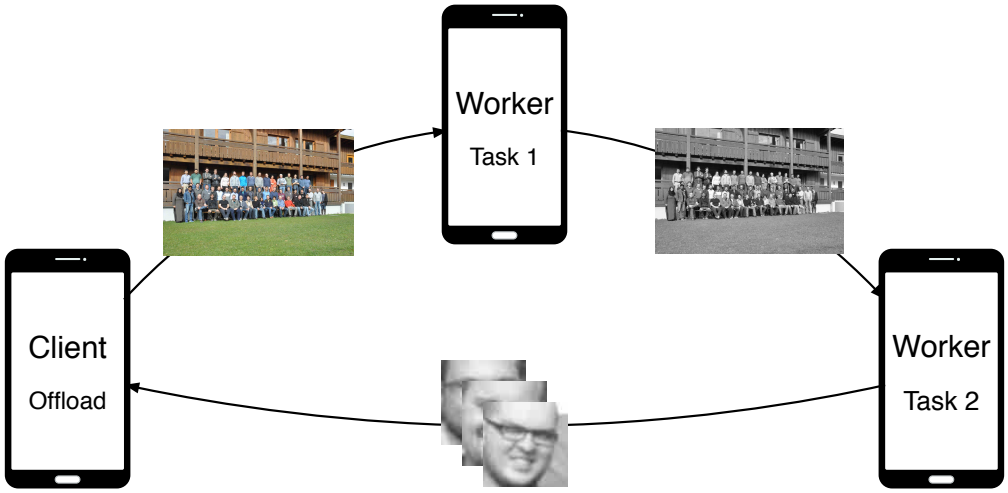


Figure 5.2: Illustrative example: executing a workflow on two workers

5.2 OPPLOAD: Offloading Computational Workflows in Opportunistic Networks

In this section, OPPLOAD is presented, a novel situation-aware framework for offloading computational workflows in opportunistic networks. Figure 5.2 shows an example for offloading a simple face detection workflow where Task 1 converts an image to grayscale, and Task 2 extracts faces and returns them to the offloading mobile device, i.e., the *client* [137]. Using OPPLOAD, mobile devices search for other suitable mobile devices, so-called *workers*, for the next task, using a novel situation-aware approach. To do so workers publish information regarding their capabilities and resources (available memory, remaining battery capacity, available hardware components, etc.). Clients perceive this information, comprehend their situation and only select capable workers based on the projected future situation. Using this method, clients can deal with incomplete information. Furthermore, workflow tasks can be executed on multiple workers that are automatically selected based on the situation of the client with respect to the network to balance the overall load, by projecting the situation change when a task is offloaded to workers, i.e., workers are rated based on their available resources, their capabilities, and the proximity to the calling client/worker. To show the feasibility of this approach, up to 30 mobile devices are emulated in different experimental settings, showing that the success rate of offloading increases by up to 40% with negligible overhead. The Python implementation[1] and all artifacts[2][3] of this section are publicly available.

This section is organized as follows. Subsection 5.2.1 presents the design of OPPLOAD and Subsection 5.2.2 covers its implementation. Subsection 5.2.3 presents an experimental evaluation of OPPLOAD and Subsection 5.2.4 discusses related work.

Parts of this section have been published previously [233].

[1] https://github.com/umr-ds/OPPLOAD
[2] https://github.com/umr-ds/OPPLOAD-experiments
[3] https://ds.mathematik.uni-marburg.de/oppload/oppload_results.tar.gz

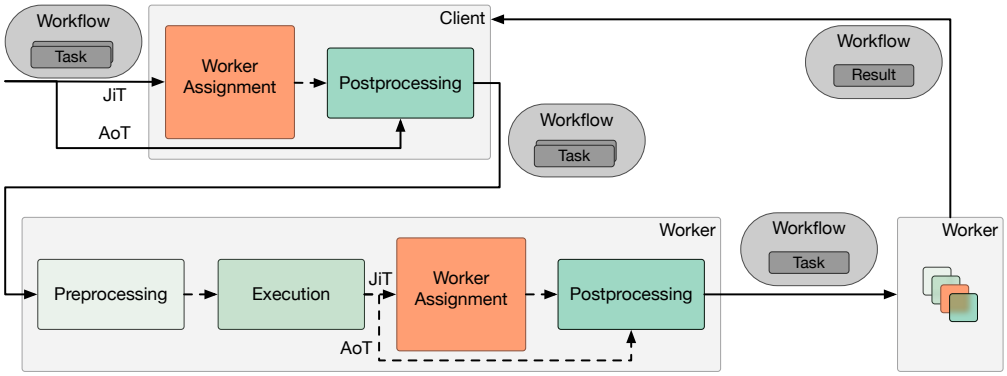


Figure 5.3: Architecture of OPPLOAD client and worker showing a possible workflow with Ahead of Time (AoT) or Just in Time (JiT) worker assignment

5.2.1 OPPLOAD's Design

Workflow-based Computations

OPPLOAD supports *workflow-based computations* where a client defines a workflow that consists of a chain of tasks. The client assigns each task to a worker, and OPPLOAD will take care of the execution order, even in unpredictable network environments. Furthermore, OPPLOAD transparently passes inputs and outputs between the different tasks of a workflow. Connectivity is achieved using protocols for Disruption-Tolerant Networking (DTN), while we assume that the communication overhead in terms of CPU and memory resources for remote execution is negligible [21].

Worker Addressing

OPPLOAD supports two worker addressing modes: *Ahead of Time (AoT)* and *Just in Time (JiT)*. This makes it possible to select the best suitable and available worker for each task, based on the user's preferences and the network environment.

Ahead of Time Using AoT addressing, the client assigns a task to a worker explicitly. It is possible to select a different worker for each task, as well as the same worker for different tasks. This mode exists mainly for two reasons. Privacy-sensitive tasks should be executed on known and trusted workers. Furthermore, worker operators might give certain guarantees, e.g., to stay in the network or to always execute a task, even under heavy load.

Just in Time In JiT mode, workers publish all services they offer periodically by broadcasting them into the network. These offers are stored on every node locally, where workers are searched from. Since in opportunistic networks nodes can appear and disappear frequently from the network, these offers are only valid for a certain time period, depending on the dynamics of the network.

If a client does not assign a task to a specific worker, OPPLOAD will transparently choose a suitable worker by passing the workflow description through a number of steps that are part of worker assignment, as shown in Figure 5.3. During the first step of worker assignment, a worker with an offer for executing a desired task will be searched in the local database. If the search is successful, the task will be executed on this worker. This mode is helpful when it is not clear whether a worker is available for a task.

Worker Capabilities

Workers announce their *capabilities* and *available resources*, such as CPU load, available memory, and other metrics. Additionally, workers announce available special hardware or other properties that help executing specific tasks better than other workers, e.g., face detection in images is more energy efficient on a GPU that may not be available on all workers. The time interval for periodic capability announcements matches the dynamics of the network. The more dynamic a network is, the more often the capabilities are broadcast. In the second step of worker assignment using JiT addressing, these capabilities are taken into account. Task requirements specified by the client are compared to the capabilities published by the workers to select capable workers.

Worker Assignment

During worker assignment, multiple capable workers may be available. Therefore, we have developed a novel *worker assignment algorithm* that distributes the workload fairly in the network on multiple workers and selects nearby and powerful workers. Instead of simply selecting a random worker or the worker with the most recent offer, we introduce a worker rating scheme based on different weighted metrics. The user has to estimate, e.g., CPU, memory, or disk space requirements for a task. Additionally, the rating scheme also keeps the tasks spatially close to the calling client. Therefore, the geographical distance between the two involved nodes is a metric of the rating. During worker assignment, the client will calculate for each capable worker how well it satisfies each requirement of a task by dividing the capabilities published by workers by the given requirements for every metric. By applying this rating scheme, the best capable worker based on the local knowledge is selected. However, this can lead to an unfair load distribution in the network, where nearby and powerful workers could be disadvantaged, since they would always be chosen. Therefore, a worker is selected from the sorted list of workers based on their rating following the folded standard normal distribution. This ensures that a nearby and powerful worker will be selected with a high probability, but the load is also distributed to different workers, leading to a fair workload distribution approach.

Error Handling

Bundle delivery in opportunistic networks cannot be guaranteed. If a worker disappears in OPPLOAD before it could execute an assigned task, the client would wait infinitely long. Therefore, users can specify a time-to-live (TTL) for a workflow. This has two implications. First, the client stops waiting for the results after the TTL has expired, making it possible to

re-issue the workflow. Second, a worker will not execute a task if the TTL is expired, which preserves resources on workers. This ensures a defined behavior in cases where no result can be retrieved in time.

If errors occur in conventional networks, clients can be notified immediately to handle the error. In opportunistic networks, this is not necessarily possible due to potentially poor network conditions. Thus, OPPLOAD handles three classes of error. The first class is a *task execution error*. These occur during the execution of the task itself. The offloaded task can implement error and exception handling on its own and provide error messages and stack traces, which OPPLOAD will deliver to the client. The second error class is a *worker selection error*. These errors occur if the execution of a task was successful, but a worker cannot find a subsequent worker during the assignment. The third error class is a *worker calling error*. These errors can occur in different situations, such as when the worker is no longer capable to execute the task or if it is not offering the service and was called by mistake in AoT mode. Error handling for these errors depends on the addressing mode. If the worker on which the error occurred was selected in JiT mode, it will inform the prior worker about the error, which will retry to assign the task to a capable worker one more time. If the second try also fails or the worker was chosen in AoT mode, the client will be informed about the error using the same communication mechanisms as before. The client is then responsible to handle the error appropriately. After an offloaded task finishes or an error occurs, OPPLOAD will clean up all involved files and bundles across all workers to save storage.

5.2.2 Implementation

We implemented OPPLOAD based on the bundle store implementation, *Rhizome*, of the Serval Mesh [97], which uses a simple epidemic DTN routing protocol. OPPLOAD is written in Python and uses Rhizome's RESTful API for handling all network-related duties. In previous work, we have conducted an in-depth evaluation of Serval in various experiments with different network setups and usage patterns [21].

Offering a Service

Workers offer a service by name, an arbitrary number of parameters, and an executable. Any executable that runs on the underlying operating system can be used, e.g., Python programs, or compiled binaries. Every worker periodically publishes the definitions of its services, and clients will then use these offers for the JiT worker assignment. In addition to the service offers, workers also announce their capabilities as key-value pairs that are published together with the service offers to reduce the network overhead.

Executing a Workflow

To execute a workflow, a user splits it into tasks to be executed across multiple workers. All tasks have to be described in a workflow description containing the desired worker (either AoT or JiT), the name of the task, and all required parameters, for each task. A workflow

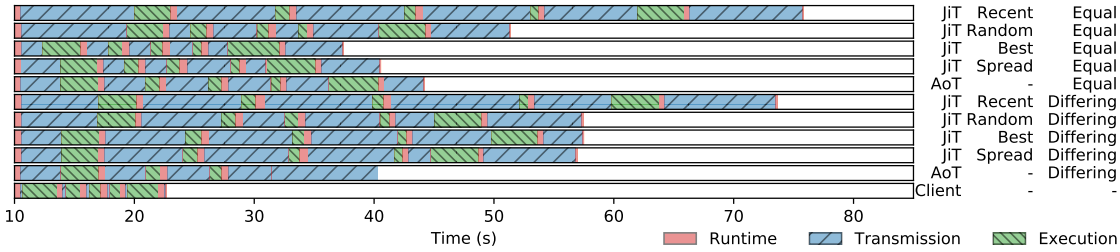


Figure 5.4: Exemplary overall workflow time in different configurations

description must include at least one task. This workflow description has to be provided to the OPPLOAD client that handles the remaining parts transparently.

A workflow description has the following form. First, a task has to be assigned to a worker, which can be an address for AoT mode or a placeholder indicating that JiT mode should be used. Then, the name of the service to be executed has to be given, followed by all parameters. Using another placeholder indicates that the output of a task should be the input for the next task. Each task can only have one result, and the placeholder is allowed only once per task. Finally, a task can have requirements that are only used during assignment for this particular task. After specifying the workflow, OPPLOAD will assign a worker to the first task, if applicable. The first step is to rank all workers, which is based on the requirements, as introduced in Section 5.2.1. For each metric, a weighted rank is calculated and summed up, using the weight and the requirements as well as the worker's capability for the particular metric. Workers are sorted based on their ranking, and a random worker is selected based on the folded normal distribution with location parameter $\mu = 0$ and scale parameter $\sigma = 1$. All files required for a task, the workflow description itself, and task results or errors will be packed into an archive that will be sent as an encrypted bundle to the selected worker. By packing everything in a single archive, fragmentation in transmission is avoided, and a worker is guaranteed to have everything required for processing the task.

When the offloaded task arrives, the worker starts preprocessing by unpacking the archive and parsing the workflow description. It will check whether it is capable of executing the assigned task, since the capabilities could have changed during the transmission due to network delays. If the worker is capable, the service will be executed. After the service finishes, the worker will replace the parameter placeholder of the next task in the description with the result of its execution. Finally, the worker assigns a next worker if required, packs everything into an archive, and passes it on.

When the final task is executed, the last worker will return the result to the client that will then trigger a network cleanup. This is achieved by having the workers remove their payloads, and it is finished when the final result is removed. If an error occurs, the worker will stop further execution, pack all intermediate files including the error log into an archive and return them as an error bundle to the client, which will raise an exception. The client is responsible to handle the exception appropriately, e.g., re-execute the workflow.

5.2.3 Evaluation

Test Setup

To evaluate OPPLOAD realistically, the network emulation framework *Common Open Research Emulator* (CORE) was used. In contrast to simulation approaches like NS-3, CORE uses Linux namespaces to execute binaries and scripts natively, which gives us the opportunity to evaluate software and frameworks as close to reality as possible by still being able to scale the experiments easily [8].

Test Cases To evaluate OPPLOAD, the algorithm of Lampe et al. [137] for detecting faces in images on smartphones was adapted. The workflow of this algorithm has five tasks. The first task is to denoise an image. The second task is to scale the image up by 10% to increase the probability of fitting a possible face into the detection window. The third task is to crop the image by 10% to decrease the image size, which speeds up the detection time while maintaining a high detection accuracy. The fourth task converts the colored image into an 8-bit grayscale image, which additionally speeds up face detection while maintaining the same detection accuracy. The fifth task detects faces on the preprocessed image. These five tasks are executed on five different workers in the network. In every experiment, the bandwidth of the network links was set to 54 Mbit/s, and a delay of 20 ms was used. All nodes were configured as workers. In JiT mode, we compared four worker assignment algorithms: (i: *recent*) selecting the worker whose offer arrived most recently, (ii: *random*) selecting a worker randomly, (iii: *best*) selecting the best available worker based on our rating, and (iv: *spread*) using the algorithm described in Section 5.2.1 to spread the load among the best available workers. Since OPPLOAD is designed for networks with mobile devices as workers, the weights for the worker rating were set to keep the tasks on nodes with high energy reserves and close to the client. Therefore, available energy and distance were weighted with 30%, CPU with 20%, and available memory and free disk space with 10%. We modelled energy using an energy unit e. These were used to model energy consumption for each task related to the task's execution time, meaning that the longer a task takes on average, the more e is consumed. A service offer from a worker was set to expire after 120 seconds, as described in Section 5.2.1. Finally, workers announced their capabilities every 2 seconds, since this is the sweet spot announcement interval, as shown by Baumgärtner et al. [22].

Baseline Evaluation

The baseline tests show how OPPLOAD performs under good network conditions. For these tests, twelve static nodes were arranged in a ring, where each node had exactly two neighbors and only the first node was a client. In AoT mode, workers were selected at the start of an experiment in the same order as they appear in the network, always skipping one node. The same workers were used for all AoT experiments for comparability. To evaluate the effect of worker capabilities, this setup was first executed with all workers equally capable of executing a task and a second time when we used the following capability distribution: 20% (2) of the workers were capable with no constraints, 40% (5) were also capable, but had less energy reserves, 30% (3) could execute the task, but with limited capabilities (like little available memory) and 10% (2) were not capable to execute the task at all. The capabilities were

Table 5.1: Average runtimes of workflow parts in the ring scenario in client-only tests and using AoT addressing

Addr.	Exec. (s)	Runt. (s)	Transm. (s)	Total (s)
Client	8.10 (0.21)	3.55 (0.11)	0.87 (-)	12.52 (0.32)
AoT	9.94 (0.26)	3.77 (0.08)	20.44 (0.13)	34.15 (0.47)

modeled using available disk space, memory, CPU resources, and energy e, which was reduced according to the above description. Since worker assignment requires randomness, our random number generator was initialized with 25 different seeds. Finally, we executed the experiments also on the client to have a benchmark for comparison.

Workflow Profiling To analyze the overhead of OPPLOAD, workflow processing was split into three phases: (i) runtime (red) of the OPPLOAD implementation, i.e., pre- and postprocessing and worker assignment in JiT tests, (ii) transmission time (blue) for transmitting the bundle, and (iii) execution time (green) of the task itself. The colors refer to Figure 5.4. The x-axis shows the workflow execution time, each bar denotes a specific configuration.

As shown in Figure 5.4, OPPLOAD does not introduce significant processing overhead. The workflows are offloaded from the clients 10 seconds after the start of the experiment. Regardless of the test configuration, postprocessing and worker assignment require about 1 second, while preprocessing can be neglected. The execution time depends on the task. While scaling, cropping, and grayscaling only require about 2 seconds, denoising and detecting faces can take up to 6 seconds.

If AoT addressing is used in known topologies, users can estimate a workflow time range in which it finishes. The downside is that if a worker is not capable of executing a task, the entire workflow will be stopped, as indicated by the second last bar in Figure 5.4. Therefore, tasks should only be explicitly assigned in cases where no other option is desirable, or if a task must be handled by a specific worker.

The major overhead is due to bundle transmission across the network. The last bar of Figure 5.4 shows the same workflow executed on the client, thus no networking is needed. The entire workflow needs about the same time as two to three tasks in the JiT tests, depending on the worker assignment. Although overhead is introduced by network related operations, it can still be better to offload workflows than executing them locally. First, the client may not be able to execute the tasks due to resource constraints or other limitations. Second, the longer the tasks take to be executed, the more negligible the communication overhead becomes. Finally, the decision whether to offload also depends on the number of hops between the offloading node and the worker, as indicated by Graubner et al. [102]. For OPPLOAD, we assume that the user decides whether to offload during the creation of the workflow.

Tables 5.1 and 5.2 show the average time needed for the parts of a workflow (the numbers in brackets show the standard deviation) in seconds. Table 5.2 indicates that the overall workflow time highly depends on the worker assignment in the JiT experiments. The recent worker assignment with an average of about 64.48 seconds requires the longest time, due to the long distance between the nodes, since their offers take longer to reach the client and thus arrive more recently. The standard deviation is also relatively high with more than 10 seconds,

Table 5.2: Average runtimes of workflow parts in the ring scenario using JiT addressing and all four assignments

Assign.	Exec. (s)	Runt. (s)	Transm. (s)	Total (s)
Recent	9.65 (0.26)	3.89 (0.09)	50.94 (10.20)	64.48 (10.50)
Random	9.82 (0.16)	3.93 (0.09)	32.60 (4.27)	46.35 (4.25)
Best	10.02 (0.28)	3.94 (0.08)	23.54 (9.63)	37.49 (9.99)
Spread	9.95 (0.20)	3.95 (0.09)	24.05 (6.82)	37.94 (7.11)

indicating long-running tasks and differing results. The random worker assignment achieves better results with about 46.35 seconds on average and a deviation of 4.25 seconds, since closer workers are chosen. Always selecting the best available worker leads to significantly lower workflow times, requiring about 37.49 seconds, but with a standard deviation of 9.99 seconds. Finally, using the spread assignment algorithm, the workflow time does not significantly differ from the previous assignment algorithm, using about 37.94 seconds, but has a better standard deviation of 7.11 seconds. If all workers are equally capable, the workflow times using the best worker or the spread algorithm do not differ. This shows clearly that in terms of workflow time, the algorithm using the best workers and our spread approach outperform the other approaches. But since not all workers are equally capable in the different capability tests, tests using the best worker have a broader standard deviation, since the capable workers are further away in the topology. This means that always using the best worker is slightly faster than using the spread algorithm, but is more unpredictable in how long the execution of a workflow will take, since the very best workers will be worn up and worse workers have to be chosen consequently. Therefore, we propose our spread algorithm as the best available solution. Executing a workflow locally at the client would only require execution time and runtime, since the networking part is not needed. As shown in Table 5.1, the total execution time is about 12.52 seconds and is pretty stable with only about 300 ms deviation. Finally, the AoT mode needs about 34.15 seconds in total and is also stable with only about 400 ms deviation. Since in AoT mode a worker is always two hops away from the next hop, the transmission is even faster than using JiT mode with the best worker assignment.

Worker Load Distribution Figure 5.5 shows the worker load distribution in all four worker assignment algorithms using JiT mode. On the y-axis, the calling nodes are shown, whereas on the x-axis the assigned worker is denoted. The lighter the color, the more often a particular client selected a particular worker.

The recent selection approach spreads the load over particular nodes, but almost always selects a worker on the opposite side of the network, leading to long-running workflows. Using a random worker, the workload is distributed on nearly all available workers. Although this leads to a fair load distribution, the profiling analysis shows that this approach does not necessarily give the fastest workflow execution times. Additionally, tasks are sent to spatially far away workers, leading to the same problems of long transmission times and network splits in mobile networks as in the recent approach. Always using the best available worker keeps the workflow execution spatially close, and the overall runtimes are the lowest achievable, but with an unfair load distribution, which disadvantages close and powerful workers over others that are also able to execute a task. In dense networks with a high offloading frequency, this could lead to overloaded nodes and empty batteries, which in the end would be less

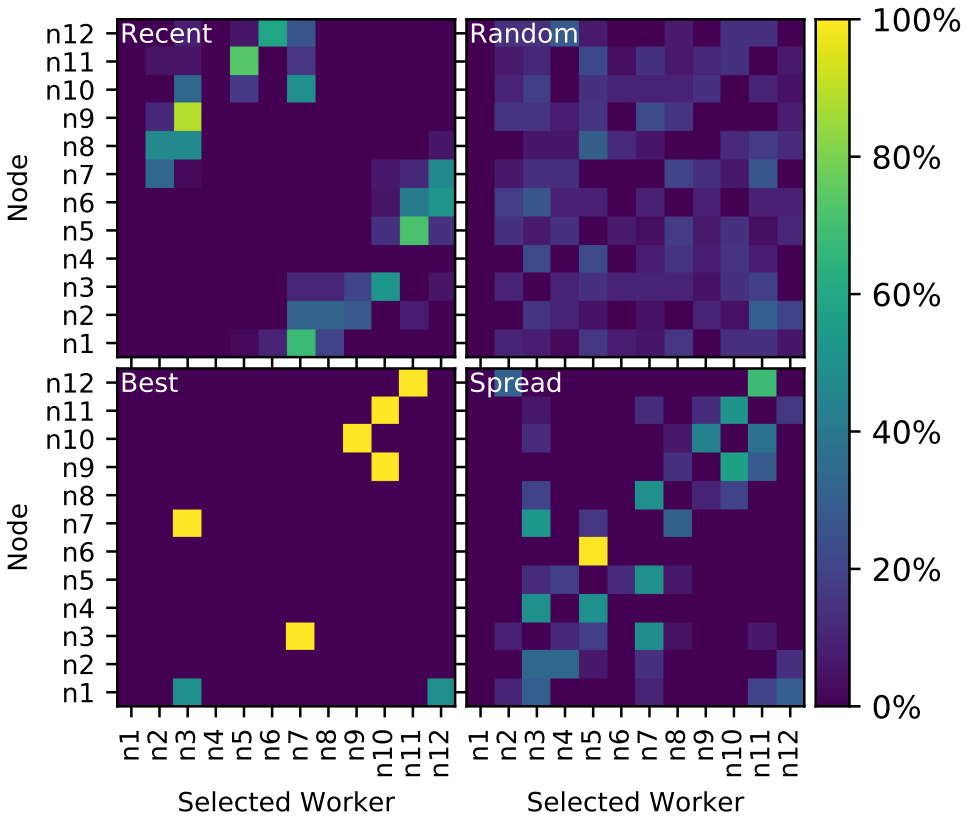


Figure 5.5: Selected workers in ring JiT scenarios

beneficial for the overall performance. Finally, our approach spreading the load on the best workers leads to the best overall results. Close and powerful workers are preferred over others, while less powerful workers also have chances to be selected. Overall, as previously shown in Table 5.2, the workflow times are nearly as good as always selecting the best worker. Thus, our algorithm should be used instead of the other presented approaches.

CPU, Memory, and Bandwidth Utilization

Figure 5.6 shows the CPU and memory utilization of an experiment in AoT mode where every worker was equally capable to execute a task. On the x-axis, the time is shown, whereas the left (blue) y-axis denotes the CPU usage and the right (orange) y-axis shows the memory allocation. In both graphs, the resource usages of all nodes are stacked, whereas 100% CPU load means that one CPU core of the emulation host is fully utilized (the emulation host had 80 CPU cores and 256 GB RAM, both are not exceeded).

During the first 10 seconds, the emulated nodes are started, configuration files are prepared, etc. After 10 seconds, Serval and OPPLOAD are started, which require many computations (e.g., loading Python interpreters into memory, computing hashes for the worker capabilities), and the CPU utilization has a high peak with more than 400% CPU. During the experiment, five peaks can be identified, which are the five tasks of the workflow. The CPU peaks are more

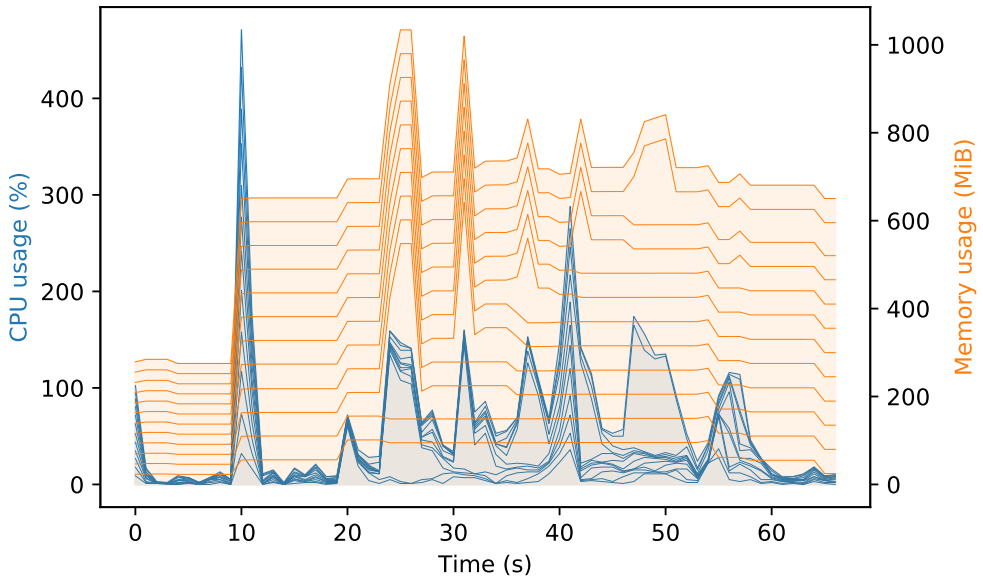


Figure 5.6: CPU and memory utilization in AoT mode; every worker capable

blurred, since not only during the task the CPU is used heavily, but also during transmitting the result to the next worker using Serval. Memory usage shows that on average every node requires about 60 MB of memory, while the execution of a task leads to peaks, due to the fact that the image and the task binary itself have to be loaded into memory.

5.2.4 Related Work

Workflow-based Approaches

To offload tasks to other mobile devices, Serendipity splits each task into smaller tasks that are offloaded if a worker is found [223]. In a mobile cloud computing scenario, Ahn et al. [7] start the execution of tasks locally and offload them to suitable cloudlets. Ravi and Peddoju [198] present an offloading algorithm where an application is partitioned into clusters containing tasks to decide whether to offload, based on a density-based clustering algorithm.

Although these proposals follow a workflow-based approach, they do not have a worker selection algorithm to distribute the load fairly in the network and/or they are not suitable for opportunistic networks.

Proximity-based, Movement-based, and Social Approaches

COMET [101] is a framework for offloading parts of applications to neighboring nodes to speed up their execution. Mtibaa et al. [174] propose a framework where a task is offloaded to mobile devices that belong to the same social situation, e.g., the same household or a group of first responders in a disaster scenario.

Wang et al. [252] present an offloading scheme for opportunistic networks where mobility patterns are analyzed to estimate the number and duration of contacts for the offloading decision. Zhang et al. [275] consider the load of a device, the availability of cloudlets, and user mobility to maximize the probability of successfully offloading tasks. Honeybee [83] includes a work sharing algorithm that employs nearby nodes to execute tasks based on job stealing.

These publications either focus on a single aspect (e.g., movement/proximity of nodes or social relationships), or they are designed for cloudlet scenarios and thus are not suitable for opportunistic networks. Additionally, most of these approaches do not follow a workflow-based approach and offload only entire tasks, without splitting them into smaller tasks.

Offloading in Cloud Environments

Deng et al. [68] decide for each task of a workflow whether it should be offloaded to the cloud or executed locally, based on the capabilities and the movement of nodes. Chatzopoulos [48] use an incentive mechanism where users have to define how many resources they are willing to spend for executing offloaded tasks. Chowdhury et al. [58] migrate tasks between cloud, mobile devices, or robots by considering energy, latency, and task execution deadlines.

All these works are designed for cloud environments and are therefore not optimized for resource savings in opportunistic networks with mobile devices.

Mobile Cloud, Edge, and Fog Environments

Fan et al. [78] present an approach where a base station in a mobile cloud scenario can either execute an offloaded task itself or further offload it to another base station. Using a fuzzy decision engine, Flores et al. [85] consider multiple criteria like CPU power to decide whether a task should be offloaded to a mobile cloud server. Yang et al. [265] offload computations in mobile cloud scenarios to maximize the throughput of applications. Chen et al. [53] formulate a game theoretic approach for offloading tasks in a mobile cloud scenario. Bellavista et al. [26] present a computation offloading approach, where tasks are offloaded to mobile edge cloud instances and the results are return over the same node or a different one, if the user has moved in the meantime. Zhang et al. [273] introduce a task allocation scheme where social sensing applications are offloaded to edge servers to maximize a node's payoff by saving energy. Yang et al. [264] propose an algorithm to offload tasks to a nearby edge server.

These approaches assume the availability of a mobile cloud, cloudlets, or similar technologies. In addition, neither worker capabilities, nor highly unreliable networks, nor workflow-based execution to preserve resources are taken into account.

Other Approaches

Funai et al. [92] present an approach that minimizes energy consumption by offloading computations across multiple hops in an ad-hoc network. Zanni et al. [272] propose an approach to split arbitrary Android apps into smaller tasks that can be offloaded. Sterz et al. [234] present a framework for remote procedure calls in disruption-tolerant networks with

separated control and data channels to cope with short contact durations. Internet-of-Things devices use more capable devices that are reachable within one hop to execute a task [73]. Feng et al. [82] present an approach where mobile devices offload tasks to other mobile devices via cellular base stations without prior knowledge of the devices' resources.

These approaches are either not suitable for opportunistic networks and faulty situations, or they only consider a very limited scope of capabilities and worker selection. Furthermore, most of them do not handle workflows but single tasks only, which is not suitable for scenarios where mobile devices are the main execution platforms.

To the best of our knowledge, there is no work that takes all these parameters into account, introduces a transparent workflow-based computational task offloading algorithm for multi-hop opportunistic networks, and provides an open source proof-of-concept implementation.

5.2.5 Summary

This section presented OPPLOAD, a novel situation-aware framework for offloading computational workflows in opportunistic networks, where workers announce their capabilities and available resources that is perceived by clients and used to comprehend their situation. Clients then can project their future situation to decide on which worker a task should be offloaded to. Experiments showed that a situation-aware approach is important for speeding up workflow execution and for spreading the load fairly on spatially close but powerful workers, which increases the rate of successful offloadings significantly. Finally, due to the workers announcing their information, clients can deal with incomplete information.

5.3 Learning Wi-Fi Connection Loss Predictions for Seamless Vertical Handovers Using Multipath TCP

In device edge computing environments, smartphones and similar mobile devices are used to provide computation and storage facilities within a network. This makes seamless connectivity desirable. The mobility of devices leads to the problem of deciding when to use which wireless connection. In this section, a novel situation-aware approach to predict Wi-Fi connection loss before the connection breaks to perform seamless vertical Wi-Fi/cellular handovers is presented. This approach relies on information perceived by multiple smartphone sensors (e.g., Wi-Fi RSSI, acceleration, compass, step counter, air pressure), followed by the comprehension of the situation followed by a projection of the mobile device's situation with respect to the state of the connection. A random forest classifier and an artificial neural network on roughly 20 GB of sensor information collected by five smartphone users over a period of three months is trained. The trained models are executed on smartphones and reliably predict Wi-Fi connection loss 15 seconds ahead of time. Experiments show that the proposed situation-aware approach provides seamless wireless connectivity, improves quality of experience by increasing mean opinion scores (MOS) from 2.7 to up to 3.8 for certain scenarios, and requires up to 50% less cellular data compared to handover approaches without Wi-Fi connection loss predictions. The data set, analysis scripts, experimental logs, and the mobile app developed in this part of the thesis are publicly available[4].

This section is organized as follows. Subsection 5.3.1 presents an overview of the approach. Subsection 5.3.2 analyzes the methods for performing predictions, and Subsection 5.3.3 evaluates the performance using vertical Wi-Fi cellular handovers on smartphones. Section 5.3.4 discusses related work.

Parts of this section have been published previously [112].

5.3.1 Conceptual Overview

Figure 5.7 shows the components of our approach and the workflow. First, raw sensor data is collected by a mobile app developed for this section and uploaded to a server for further processing. The raw data is appropriately preprocessed and enriched with additional higher level features. The resulting data is then used to train and evaluate a random forest classifier and different neural network architectures. The data preprocessing operations as well as the trained models are transpiled to Java code and integrated into the mobile app on the smartphone, which in turn makes online predictions for Wi-Fi connection loss 15 seconds ahead of time. Based on these predictions, vertical Wi-Fi/cellular handover is performed using MPTCP. We explain the main steps of our approach in more detail below. Neural network model building and Wi-Fi connection loss predictions for performing MPTCP handovers on a smartphone are discussed in Sections 5.3.2 and 5.3.3, respectively.

[4] https://umr-ds.github.io/seamcon/

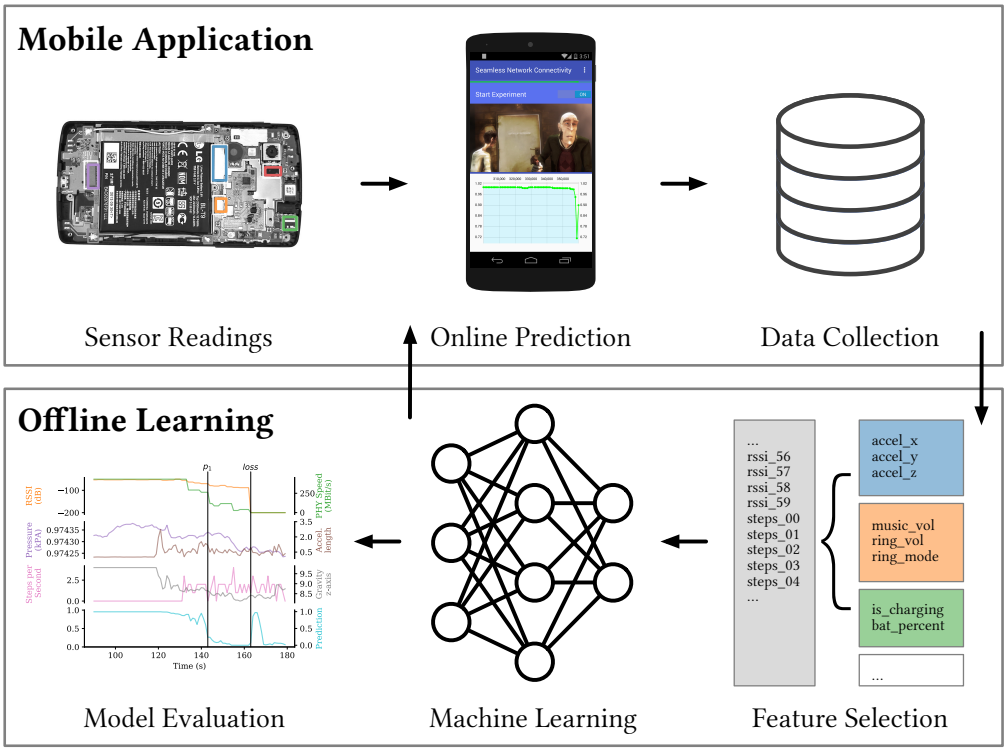


Figure 5.7: Mobile application and offline learning

Smartphone Sensors

Modern smartphones offer a variety of sensors that directly or indirectly measure different properties, as explained below. Even though every individual feature might not be a good Wi-Fi connection loss indicator, combinations of seemingly irrelevant features can improve the prediction accuracy.

Motion Depending on the abstraction level, direct motion sensor readings (e.g., accelerometer, gyroscope, magnetometer), sensor readings cleaned from unwanted influences (e.g., gravity, linear acceleration, rotation vector), or higher level sensor readings as hardware processed triggers (e.g., significant motion, step counter, step detection), are good predictors for user movement.

Orientation Orientation sensors can reveal more specific situations, where a phone is in the pocket or laying on a table. The proximity sensor is typically used to detect whether the smartphone is held to the ear, but can also be a good hint for other situations, e.g., to detect whether the smartphone is face down on the table.

Environment Environmental sensors include sensors for measuring ambient light to control screen brightness, humidity, air pressure, and ambient temperature. Rapid changes in these sensor readings can reveal a sudden change of the smartphone situation, e.g., going outdoors.

Global Position GPS can be useful in combination with a world map of Wi-Fi availability. Due to quality concerns with indoor GPS traces and high energy consumption, we discarded GPS for this approach.

User Interactivity The user's current situation can be derived from various indicators, like device state (interactive, idle, power save), current charging state, audio state (speaker, headphones, and their volumes), and ringer mode.

Wi-Fi Properties Wi-Fi properties, obtained by the radio interface, provide insights into the current connection quality along with reachable networks. Relevant indicators include RSSI, data link layer speed, and used frequency bands.

Sensor Data Preprocessing

To learn Wi-Fi connection loss predictions, the sensor data needs to be preprocessed. The time component of the sensor readings needs to be incorporated in the feature vector.

Sensor Sampling The used heterogeneous sensors have different reading frequencies. Motion and orientation sensors can be read with a rather high sampling rate R of 50 Hz, while other sensors are available and useful just under 1 Hz. As a trade-off between energy consumption and sensor data quality, we chose a sensor data sampling rate of $R = 1$ sample per second. Sensors with lower sampling rates are filled until a new value becomes available.

Observation and Prediction Window To enrich the discrete sensor readings and to consider the temporal component, the sensor readings are processed in an observation window OW. We use an observation window of 60 seconds, which is derived from common walking speeds and Wi-Fi access point ranges. The earlier a Wi-Fi connection loss is predicted, the more effective the transition between Wi-Fi/cellular is. To define an upper bound on the prediction window, the quality characteristics of the used network protocols are important. Transport protocols, such as TCP, use slow-start to avoid congestion. To compensate for this low-bandwidth start, an early prediction is useful. As a trade-off between performance and farsightedness, and to avoid long-running redundant MPTCP connections, which are energy and data plan consuming, we use a prediction window of up to 15 seconds.

Feature Vector The feature vector presented to the learning algorithm consists of the sensor readings in the observation window. Each individual sensor contributes $OW \times SR$ values to the feature vector. In the selected configuration, this results in 60 values per sensor. Furthermore, all features are normalized by removing the mean and scaling to unit variance, as required for the machine learning algorithms used in our approach.

Precision and Recall

When the Wi-Fi connection loss is predicted too early or too often, this can result in higher consumption of restricted data plans of the users. Predicting it too late, on the other hand, can result in a bad QoE. The primary goal is to reach a high recall in predicting Wi-Fi connection loss. In terms of energy efficiency, the secondary goal is to reach a high precision predicting Wi-Fi connection loss, thus not performing unnecessary handovers.

5.3.2 Learning Wi-Fi Loss Predictions

In this section, our novel data-driven approach to predict Wi-Fi connection loss is presented. Feature vector preparation and two sets of features considered will be presented. Finally, a data example and the machine learning results will be discussed.

Data Set

We collected about 20 GB of smartphone sensor data from 5 users, with more than 900,000 unique samples, over a period of three months. The users were advised to let the mobile application run throughout the day, thus the traces contain data from the users' daily lives.

Training and Test Set Machine learning methods require separate data sets for training and testing to verify the generalization abilities of the models. We investigated different ways of building training and test sets: (i) randomly split the available samples into, e.g., 70% training and 30% test data, and (ii) split by users.

Feature Vectors

All features collected on the smartphones can be used as predictors for Wi-Fi connection loss. We used two feature vector sets, namely the *Full* and the *Reduced Feature Vector*.

Full Feature Vector The data collected by the different users shows that some features are not available on all devices. The 25 features selected for the full feature vector consist of values of all available sensors: Atmospheric pressure: x, $delta$; Linear acceleration: x, y, z, $length$; Step counter: $delta$; Power: $is\ charging$, $battery\ percentage$; Gravity: x, y, z; Gyroscope: $length$; Magnetic field: x, y, z; Orientation: x, y, z; Rotation: x, y, z; Wi-Fi: $frequency$, $speed$, $RSSI$. Thus, the feature vector consists of $25 \times 60 = 1500$ features (i.e., with a 60 seconds observation window).

Reduced Feature Vector Many of the sensors, like linear acceleration and gyroscope, described in Section 5.3.1 share underlying features due to their physical properties. The number of sensors can be reduced by leaving them aside. For the *Reduced Feature Vector*, we used the following sensors: Atmospheric pressure: $delta$; Linear acceleration: $length$; Step counter: $delta$; Power: $is\ charging$; Gravity: z; Wi-Fi: $frequency$, $speed$, $RSSI$.

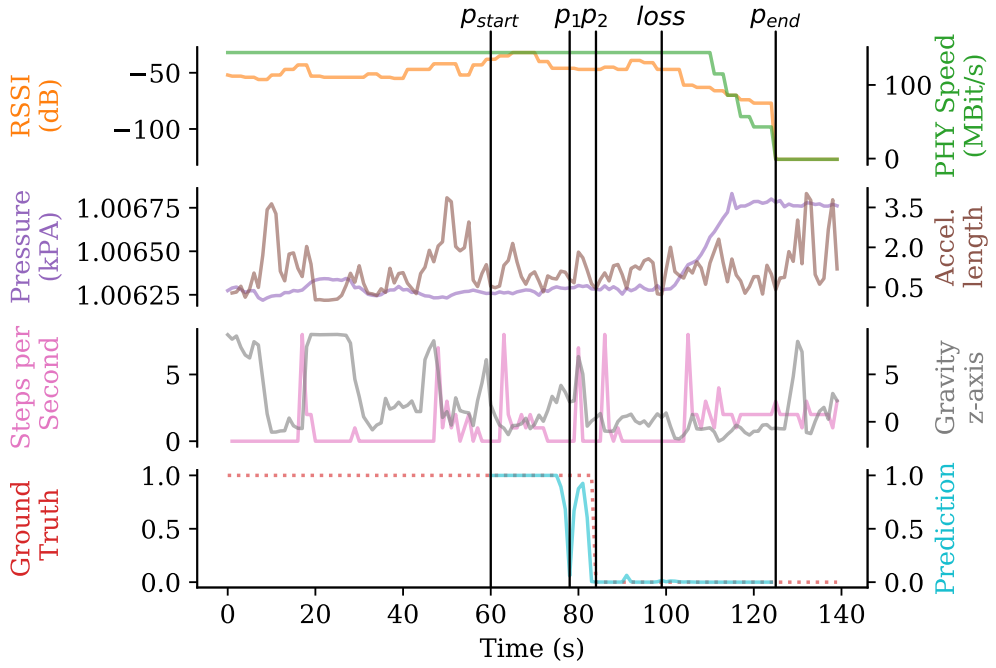


Figure 5.8: Different sensors leading to an early (p_1) and an ideal (p_2) prediction of Wi-Fi connection loss, based on a trained model with randomly split data

Sensor Data Example

Figure 5.8 shows an example of several sensor data values collected by a smartphone. The figure shows the computed ground truth and a prediction probability value of a neural network based on the *Full Feature Vector*, i.e., a probability value $< 50\%$ means that a Wi-Fi connection loss is predicted and vice versa. The graphical representation of the sensor values shows that no obvious correlation between one of the sensors and the prediction ground truth exists. Nevertheless, each of the sensors shows some information that could be useful. For example, the atmospheric pressure sensor rises from $t = 100$ to $t = 115$, which could be caused by changing the floor in order to leave the building or by a changing ventilation. In combination with the step counter delta, the first option is more likely, also resulting in a higher likelihood for a Wi-Fi connection loss. Another example is the gravity sensor's z axis that reports about 9.81 for the time period from $t = 20$ until $t = 35$, which together with the linear acceleration sensor is a good sign for laying flat on a table. This again reduces the likelihood of a Wi-Fi connection loss event.

For the neural network shown on the bottom in Figure 5.8, a 60 seconds observation window has to be filled before the first prediction is performed at p_{start}. The classification ends at p_{end}, since the operating system reports that Wi-Fi is unavailable. Since Wi-Fi becomes unavailable at $loss$, the ground truth is 0 from p_2 ongoing, matching the 15 seconds prediction window. The neural network classifier matches the ground truth quite well, except for p_1, where the classifier predicts the loss slightly too early. This example shows that the combination of sensors available on today's smartphones can lead to an effective prediction of Wi-Fi connection loss.

Table 5.3: *Reduced Feature Vector*, randomly split data, different learners and configurations

Metric	Forest	*NN 1*	*NN 2*	*NN 3*
Loss Prec.	0.89	0.95	0.97	0.97
Loss Recall	0.98	0.94	0.95	0.95
F_1-score	0.93	0.94	0.96	0.96

Machine Learning Results

This section presents results of training different methods with the data to predict Wi-Fi connection loss: (i) a random forest classifier [146], and (ii) a multi-layer neural network. In particular, we use the MLPClassifier and RandomForest implementations of scikit-learn [189].

Random Forest Since random forest learning depends on equally distributed samples, the data is down-sampled accordingly to match this criterion. The random forest consists of 10 random trees, learned to use the Gini criterion. The overall performance of the random forest is satisfactory, since all values are greater than 0.97. However, the precision of the Wi-Fi connection loss class was not very high (0.86), ultimately resulting in triggering early or unnecessary handovers.

RSSI-only Neural Network Another basic learning approach is to limit the learner to only use the timeseries of RSSI values, as presented in Section 5.3.4. During our experiments, different configurations of the neural network were evaluated. The overall performance is comparable to the performance presented in the related work. The classification quality of the Wi-Fi connection loss class did not exceed an F_1-score of 0.95.

Random Data Split The results for neural networks learned with randomly split data depend on the neural network architectures. Table 5.3 provides an overview of different classifier approaches with the *Reduced Feature Vector*. Classifier *NN 1* consists of 100 hidden neurons, *NN 2* of (300, 200, 100) neurons, and *NN 3* of 5 hidden layers containing (400, 400, 400, 400, 400) neurons. All results were achieved using 70% of the data set exclusively for learning and the remaining 30% for testing. In our experiments, *NN 1* can reach a classification quality comparable to the random forest classifier. The F_1-score of the Wi-Fi connection loss class reaches up to 0.94, with either a high precision or a high recall, but never both. In general, the negative class, representing stable Wi-Fi connections, is predicted well by all tested neural network classifiers. The experiments show that neural networks can reach both high precision and high recall in the positive Wi-Fi connection loss class.

The results presented in Table 5.3 show that *NN 2* and *NN 3* provide reasonably good performance for both precision and recall in the Wi-Fi connection loss class. Even the neural network *NN 2* consisting of three layers shows significant improvements compared to the flat neural network discussed in the previous paragraph. It reaches an F_1-score of 0.96 with slightly lower recall or precision.

Neural network architectures with up to 10 hidden layers were tested. Both precision and recall could not be improved. Splitting the data randomly, *NN 2* and *NN 3* perform equally and enable a prediction with 97% precision, 95% recall, and a combined F_1-score of 0.96.

User-based Data Split When testing for previously unseen users, the precision of the loss worsens in our prediction. With 0.93, 0.92, and 0.79 precision in the Wi-Fi loss class, the *Reduced Feature Vector* generalizes better compared to the *Full Feature Vector* resulting in 0.91, 0.72, and 0.68 precision.

The results show that the neural networks are capable of generalizing even among different users and devices. A good classification can be achieved using a neural network with the *Reduced Feature Vector*. Providing a reasonably well basic functionality in the starting phase, with data collected on the device, the classification can be improved during usage.

For further application evaluation, the *Reduced Feature Vector NN 3* model was used.

5.3.3 Improved Video Streaming with Seamless MPTCP Handovers

As presented in Section 5.3.2, the neural network models reliably predict Wi-Fi connection loss. To show the usefulness of these results, we evaluated the performance when performing handovers in real-world mobile usage scenarios. In the following, we present a seamless Wi-Fi/cellular handover during DASH video streaming sessions.

Seamless Network Connectivity App

To gather the training data, perform the prediction, and test the applicability of the approach, we implemented a mobile application that performs the following tasks:

Sensor Data Collection and Preprocessing The sensor readings described in Section 5.3.1 are cached in memory and written periodically to a local SQLite database on the smartphone. When a run ends, the database is uploaded to a server. To execute the neural network on the smartphone, the sensor values are preprocessed similarly to the offline learning process. The mean, variance, and the observation window determined offline are used.

Online Prediction The offline learned neural networks are transpiled to Java using the sklearn-porter[172] framework, which allows execution of the same neural networks trained with sklearn on the device. This execution on the Android device allows us to achieve low delays in predictions, independence of Internet access, and protects user privacy.

Demonstration & Reporting We demonstrate the feasibility of the proposed approach using an embedded DASH video playback functionality. Here, the goal is to highlight potential in improved playback quality and stability made possible by seamless connectivity. We use the open movie Elephants Dream, streamed from a server in the university network. The video was pre-encoded using the h.264 encoder for video and AAC for audio, in three bandwidths 1, 2 and 4 MBit/s and a segment length of 2 seconds. For video playback, the JavaScript-based DASH.js player (v 2.5.0) was used with a buffer size of 10 seconds in conjunction with the BOLA adaptation algorithm. To analyze the QoE, we collect and report raw video metrics in each streaming session while the video is playing to the server including stalls, playback bit rates, quality adaptations, and buffer levels for later evaluation.

MPTCP Handovers We use the Wi-Fi connection loss prediction to trigger the cellular subflow establishment for MPTCP *before* the Wi-Fi connection is lost. We implemented our approach on top of the MPTCP kernel implementation for Android[5] and the *MultipathControl*[6] app of De Coninck et al. [61]. Furthermore, the video server uses MPTCP version 0.92 with the redundant scheduler and the fullmesh path manager enabled.

Experimental Setup

Our experiments consist of 3 connectivity modes in 4 scenarios each performed 5 times, resulting in a total of 60 iterations. The experiments were performed on a Google Nexus 5 smartphone running a rooted Android and the MPTCP Kernel version 0.89.5. The following connectivity modes were evaluated:

- *Stock* Android: The default Android mechanism was used to detect Wi-Fi unavailability. No transition mechanism was used to have a baseline to compare with.
- *MPTCP*: To see how MPTCP can improve handovers, it was enabled for the entire run in these tests. The cellular uplink was used as the second interface, and both client and server used the default scheduler.
- *Seamless*: During these tests, the *Reduced Feature Vector* neural network in configuration *NN 3* was used, since it showed the most promising results. MPTCP is enabled when a Wi-Fi loss is predicted and disabled when Wi-Fi is available and no loss is predicted for 5 seconds.

The following routes are chosen to evaluate scenarios in which Wi-Fi connection losses can occur. Figure 5.9 shows the room plan of the university building.

Scenario 1: Leaving the Office Starting in the office, the smartphone is connected to the office Wi-Fi. The tester leaves after 120 seconds of video playback and heads towards the exit of the building. After the Wi-Fi connection is lost (determined in advance, roughly 50 meters) the tester waits for 10 seconds and ends the scenario.

[5] https://multipath-tcp.org/pmwiki.php/Users/Android
[6] https://github.com/MPTCP-smartphone-thesis/MultipathControl

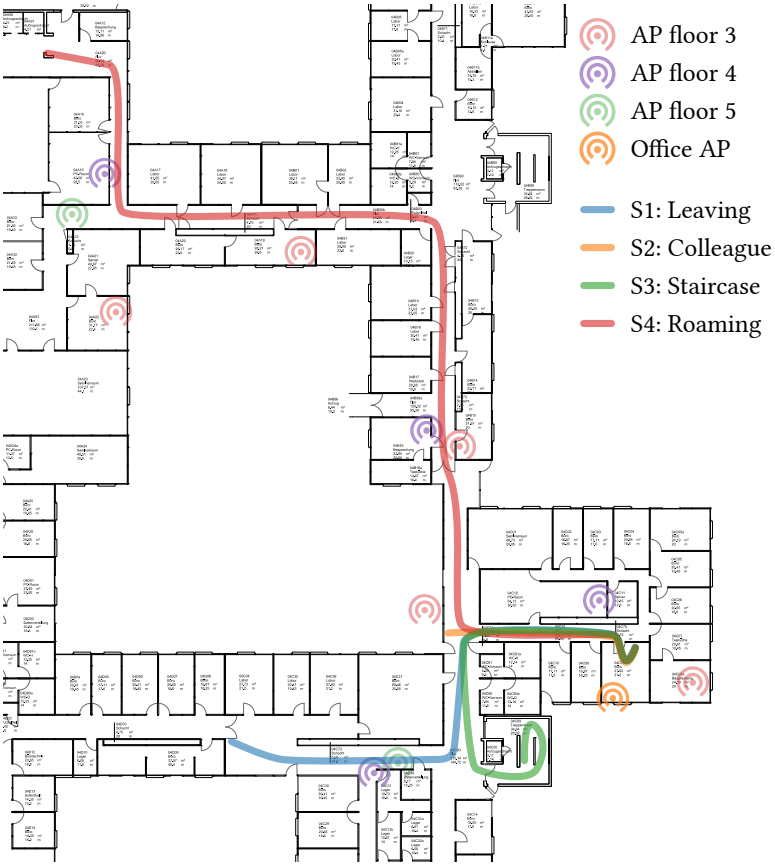


Figure 5.9: Map with Wi-Fi APs and scenarios routes

Scenario 2: Visiting a Colleague The beginning is similar to *Scenario 1*, but the tester walks around about 20 meters away from the office, visiting a colleague, but not leaving the Wi-Fi range. The tester stays for 10 seconds and then walks back to the office.

Scenario 3: Using the Staircase Starting as before, the tester leaves the office on the same route, but then uses the staircase to go up one floor and stays there for 10 seconds. The scenario shows the impact of a Wi-Fi connection that is available but not usable.

Scenario 4: Wi-Fi Roaming Support Starting in the office, the device is connected to the university network. The tester leaves after 120 seconds and heads towards the other end of the building, roaming between multiple possible Wi-Fi APs shown in Figure 5.9. The tester stays near the exit for 10 seconds and then walks back the same route. This scenario is created to further investigate the support of roaming gaps in corporate wireless networks where roaming might be available, but is not sufficient to achieve a high QoE.

Measuring Quality of Experience

The DASH video streaming technology is widely available, used by many vendors, and evaluated well. To measure the perceived QoE, several technical values are captured that are used to compute mean opinion scores (MOS) [238], as discussed below.

Direct Measurements During the experiments, the DASH video player reports different technical parameters to a server. At the beginning of a video, the initial buffer has to be filled. This takes time, resulting in *initial stallings* that are perceived to be more disturbing the longer they take. Furthermore, *stalling events* during the video are also reported. Apart from the stallings, the *number of adaptations* is counted, since many adaptations also negatively influence the QoE. In addition, the percentage of *time spent in the highest achieved quality* is measured. From a user's perspective, it is better to hold a certain quality as long as possible, even if it is not the best quality available. Since stalling events and quality adaptations partly depend on the buffer level (i.e., how much playable video is in the buffer), the *buffer level* is also captured. The buffer level should be as constant as possible for about 10 seconds.

Finally, a packet dump is performed on the server to allow further analysis of the connections created by MPTCP.

QoE Metrics Apart from directly evaluating the metrics discussed above, derived metrics are used to capture relations between these metrics and their impact on QoE. The QoE_{stall} (Equation (1)) is derived on a MOS scale (where 1 denotes a bad user experience and 5 an excellent one) based on the stalling durations and frequencies during video playback. Furthermore, $MOS_{quality}$ (Equation (2)) is deduced based on playtime in the highest achieved quality (t). L denotes the average length of all conducted stallings (initially or during video playback) and N the number of stallings, again either initially or during playback. Since this approach focuses on Wi-Fi connection loss events, we do not evaluate initial stallings.

$$MOS_{stall} = 3.5 \times e^{-(0.15 \times L + 0.19) \times N} + 1.5 \tag{5.1}$$

$$MOS_{quality} = 0.003 \times e^{0.064 \times t \times 100} + 2.498 \tag{5.2}$$

$$MOS_{combined} = \frac{MOS_{stall} + MOS_{quality}}{2} \tag{5.3}$$

Finally, Stohr et al. [238] propose the average MOS, denoted as $MOS_{combined}$ (Equation (3)), denoting a total user perception not only depending on stalling or quality adaptations. We use $MOS_{combined}$ to evaluate QoE.

QoE Experimental Results In Table 5.4, the overall results of the performed tests are presented, namely the number of stalling events (# St.) and the average duration of a stalling event (∅ St.), the number of adaptations (# A.), the relative time in the highest playback quality (HQ), and the average transmitted data (∅ TD).

Table 5.4: Overview of experimental results

(a) Scenario 1: Leaving

Mode	# St.	∅ St.	# A.	HQ	∅ TD
Stock	3	1.46 s	23	87 %	21.75 MB
MPTCP	0	0 s	20	89 %	41.32 MB
Seamless	0	0 s	27	88 %	36.11 MB

(b) Scenario 2: Colleague

Mode	# St.	∅ St.	# A.	HQ	∅ TD
Stock	0	0 s	10	92 %	0 MB
MPTCP	0	0 s	10	91 %	9.98 MB
Seamless	0	0 s	17	92 %	9.59 MB

(c) Scenario 3: Staircase

Mode	# St.	∅ St.	# A.	HQ	∅ TD
Stock	3	2.06 s	49	80 %	0 MB
MPTCP	0	0 s	32	87 %	33.92 MB
Seamless	0	0 s	28	85 %	16.81 MB

(d) Scenario 4: Wi-Fi Roaming

Mode	# St.	∅ St.	# A.	HQ	∅ TD
Stock	18	14.98 s	42	53 %	0.89 MB
MPTCP	0	0 s	38	86 %	71.99 MB
Seamless	15	5.47 s	23	84 %	15.50 MB

Scenario 1 As shown in Table 5.4a, the *Stock* tests performed worst with 3 stalling events in total and an average stalling duration of about 1.5 seconds, while neither *MPTCP* nor *Seamless* tests did show any stalling events, which is a significant improvement compared to the stock tests. The amount of transferred data over cellular is high in the *MPTCP* test and low in the *Stock* test. *Seamless* results are between these two tests, thus saving cellular data compared to MPTCP, while still avoiding stallings. The results of these tests show that our prediction can avoid the handover gap completely.

When looking at the buffer levels, video stream quality and the used bandwidth, it can be seen that based on the prediction of *Seamless*, the cellular subflow is established proactively, resulting in a seamless handover and thus no video stalling.

Apart from improvements of these technical values, our approach improves QoE for users, as expressed in the $MOS_{combined}$. Figure 5.10 shows the $MOS_{combined}$ on the y-axis and the different connectivity modes on the x-axis, grouped by scenario. For the stock tests, the $MOS_{combined}$ is between about 2.5 (poor) and 3.5 (fair), indicating that the playback is not totally unsatisfactory, but far away from a great experience. *Seamless*, on the other hand, achieves a $MOS_{combined}$ of almost 4, indicating a good QoE, as high as in *MPTCP* tests.

Scenario 2 As shown in Table 5.4b, all tests are comparable for all metrics, showing that our approach does not introduce any negative effects in already good situations. The transferred amount of data over cellular in *Seamless* is about as high as in the *MPTCP* tests. This is

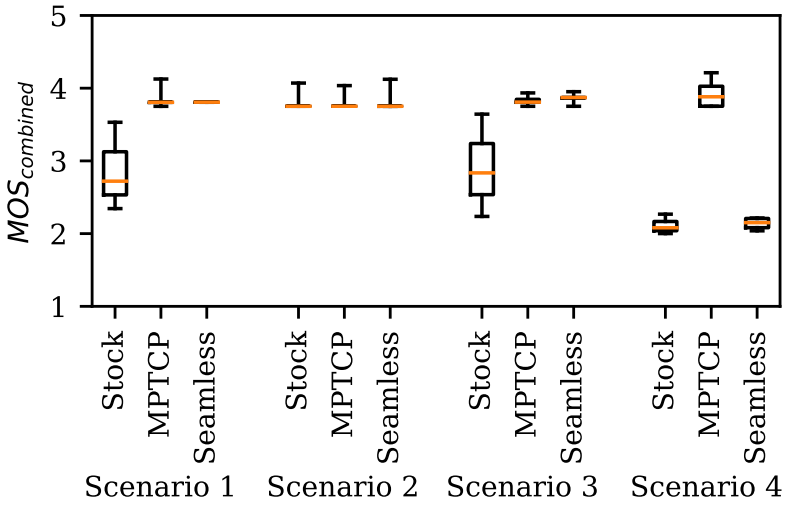
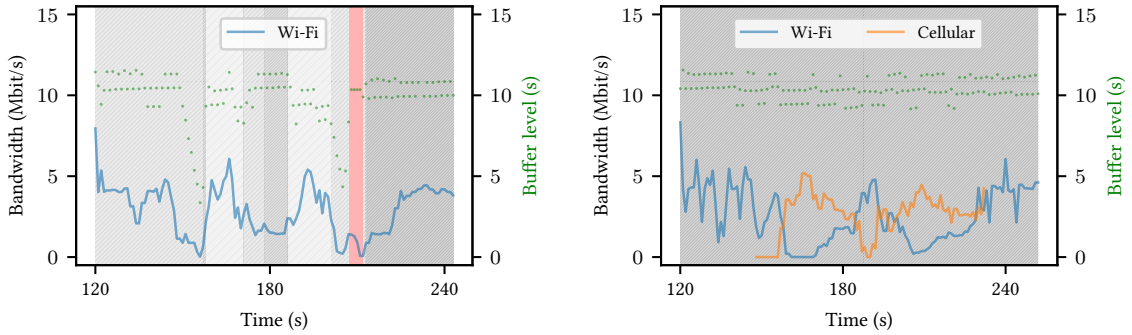


Figure 5.10: $MOS_{combined}$ values grouped to connectivity modes and scenarios.



(a) Stock Android **(b)** Seamless

Figure 5.11: *Stock* and *Seamless* in *Scenario 3*

because the classifier predicts a Wi-Fi connection loss due to the movement of the smartphone and thus switches to the cellular network, even though this is not necessary.

Neither the technical metrics like buffer level or used bandwidth, nor the MOS values differ in the these experiments, thus they are not further evaluated here, again indicating that our approach does not worsen the situation by any means.

Scenario 3 As shown in Table 5.4c, the stock tests performed worst with 3 stallings and an average stalling time of about 2 seconds. Additionally, with 49 adaptations and only 80% of the time at the highest achieved quality, the stock tests perform badly. *MPTCP* and *Seamless* do not stall at all. With 28 and 30 adaptations and 85% of the time at the highest achieved quality, the results of our approach are as good as in the *MPTCP* tests, again showing significant improvements over the stock implementation. The data usage over cellular shows the same behavior as in *Scenario 1*.

Figures 5.11a and 5.11b show bandwidth and buffer level for *Scenario 3*. In the stock tests, the maximum distance is shown in the used bandwidth around seconds 150 and 210. *Seamless*

improves this situation and establishes a cellular connection timely resulting in no stallings. $MOS_{combined}$ during the stock tests shows again a relatively bad QoE with about 2.5 to 3.5 compared to the high MOS values of about 4 during MPTCP and *seamless* tests.

Scenario 4 The results of the stock tests in Table 5.4d indicate that Android does not handle *Scenario 4* well. The video stalls 18 times and for about 15 seconds on average. The video quality adapts 42 times in total and stays only for 53% of the time at the highest achieved quality. MPTCP, on the other hand, handles *Scenario 4* very well with no stallings, few quality adaptations, and 86% of the time at the highest achieved quality.

Although *Seamless* cannot completely cope with the situation, the results are much better than in the stock tests. With 15 stallings and an average stalling duration of 5.5 seconds, just 23 quality adaptations and 84% percent of the playtime at the highest achievable quality, the results indicate an improved QoE using our approach. The benefits of these improvements come with the cost of using more data (14.61 MB) over the cellular network, but only 21.53% of cellular data compared to the MPTCP tests.

It is evident that our approach predicts Wi-Fi connection loss correctly, since connection establishment occurs in a timely manner. However, the cellular interface does not reach the high bandwidth used by MPTCP in the same scenario, which might be due to the fact that the cellular interface requires a longer starting phase in the concrete area of the building. Also, due to the relatively short Wi-Fi-less gaps investigated in this scenario, the cellular connections are dismantled shortly after they are established. A model optimized not only for predicting Wi-Fi connection loss events but also Wi-Fi recovery could improve such scenarios by keeping the cellular link longer alive.

$MOS_{combined}$ during the *Seamless* and stock tests is comparably bad with a value of about 2, while MPTCP still reaches a MOS of about 4. Nevertheless, our approach reaches a slightly higher QoE than stock Android, as shown in Figure 5.10.

5.3.4 Related Work

Predicting Wi-Fi Connection Loss

Several approaches to predict Wi-Fi connection loss for performing handovers have been proposed in the literature [5]. Nasser et al. [178] use neural networks to predict Wi-Fi connection loss events based on RSSI. Horich et al. [114] use a fuzzy logic controller (FLC) for making decisions about performing handovers, where the parameters for the FLC are learned to use a neural network.

Lin et al. [147] propose to use standard Wi-Fi connection properties and a neural network to predict Wi-Fi connection loss. Monsour et al. [158] use a combination of user velocity and the Allan variance of the RSSI to predict Wi-Fi connection loss, and use PMIPV6 to manage the predicted Wi-Fi connection loss. Khan et al. [128] propose a fuzzy logic system to predict Wi-Fi loss events based on various parameters, such as delay, jitter, bit error rate, packet loss, communication cost, response time, and network load.

These approaches are limited to information of wireless connections, which may be helpful to create metrics for Wi-Fi quality, but is not always the best information for predicting Wi-Fi connection loss. In contrast, our approach considers information from a wide range of smartphone sensors that indicate the usage context, leading to high-quality predictions. Other approaches incorporate the mobility of the users [182] or higher level features like social group affiliation, time-of-day, and average duration a user spends in a particular network [251].

The predictions in all of these approaches depend on external factors. In contrast, our approach only requires information that every current mobile device provides and thus can be used in a straightforward, economic attractive manner. To best of our knowledge, there is no work that uses smartphone sensor data to predict Wi-Fi connection loss.

Performing Vertical Handovers

There are extensions to the traditional Internet Protocol that allow users to keep a session alive when (vertical) handovers are performed [192]. These approaches are based on home and foreign agents that forward traffic for the mobile host. Although they are around for a long time, mobile IP is not supported widely. Ma et al. [154] propose a vertical handover method based on the Stream Control Transmission Protocol. While the proposed method is network-independent and thus does not require home and foreign agents, traditional TCP-based applications cannot benefit from the advancements. MPTCP is a TCP extension supporting multiple subflows for a single TCP connection [87]. MPTCP improves throughput and reliability in data center and mobile environments [54, 197]. Paasch et al. [185] evaluate MPTCP as a vertical handover mechanism. The authors propose three MPTCP modes for handover scenarios, namely *Full*, *Backup*, and *Single-Path Mode*. The first two modes maintain subflows on all interfaces, while the *Single-Path Mode* exploits the break-before-make design of MPTCP. Pluntke et al. [195] use MPTCP as a vertical handover mechanism to shift connections between cellular and Wi-Fi connectivity and finally to save energy. De Coninck and Bonaventure [65] futher improve the handover by speeding up packet retransmissions after the cellular subflow is established.

The handover mechanisms in these approaches are either reactive, resulting in temporary connection losses, or use redundancy, leading to high bandwidth consumption, which is often contrary to the users' preferences.

5.3.5 Summary

In this section, a novel situation-aware approach to predict Wi-Fi connection loss to perform seamless vertical Wi-Fi/cellular handovers was presented. The approach is based on perceived sensor information on mobile devices. We demonstrated that the trained neural networks reliably predict Wi-Fi connection loss 15 seconds ahead of time when users move around, with a precision of up to 0.97 and a recall of up to 0.98. Using this situation-aware approach, a mobile device can comprehend its situation with respect to the connection status and successfully predict whether the connection is lost. This leads to the ability to decide whether a second connection should be established to provide seamless connections in device edge computing environments.

5.4 Summary

This chapter presented contributions to the novel approach of situation-aware device edge computing. It was shown how mobile devices can leverage situation-awareness to decide which devices should be used to execute a process within the same network and how to predict whether the wireless connection of a mobile device is about to break so that it can be decided if a second connection should be established.

In particular, the following contributions were presented:

- *OPPLOAD:* A novel situation-aware framework that uses information regarding the capabilities and limitations of other mobile devices to decide on which device a process should be offloaded to.
- *Seamless Handovers:* Using information provided by sensors of mobile devices to derive the own situation regarding the connection status for deciding whether a second connection using a different wireless technology should be established in cases of projected connection losses.

6

Situation-aware Embedded Edge Computing

In this chapter, two contributions to the field of situation-aware embedded edge computing are presented. Section 6.1 motivates why situation-awareness is important in the area of embedded edge computing. In Section 6.2, an innovative programming language is presented that follows the reactive programming paradigm and facilitates novel situation-aware applications executed on embedded devices. In addition, Section 6.3 presents an approach that uses a novel Complex Event Processing (CEP) engine to enable embedded devices to execute CEP queries to support analysis of sensor information so that situation-awareness can be achieved. Section 6.4 concludes this chapter.

Parts of this chapter have been published previously [104, 235].

6.1 Motivation

As discussed in Chapters 4 and 5, infrastructure edge computing and device edge computing are mainly concerted with providing computation and storage resources in the proximity of users, so that latencies are reduced, quality of experience is increased and, in case of device edge computing, these resources are even available when infrastructure edge computing or cloud computing facilities are not available. In the area of embedded edge computing, however, the primary concern is to reduce the amount of data in the network, since especially IoT and sensor devices continuously produce data that is pushed into the network. Transmitting these amounts of data can lead to network congestion and high latencies during the evaluation of sensor data. Thus, pre-processing, analyzing, and aggregating the data directly on the embedded devices reduces the network load and immediately provides derived information [127, 221]. To support situation-awareness, however, the predominantly used programming and analysis paradigms are not well suited. The vast majority of approaches are based on languages typical for these platforms, such as C/C++. APIs and SDKs are then used to implement the aggregations and analyzes that are supposed to cover the weaknesses of these languages, e.g., error-prone memory management, weak type systems complicating compile-time optimizations, etc. Furthermore, these languages follow the paradigms of imperative and procedural programming. The consequence is that data management and the resulting call graphs become confusing, error-prone, and opaque. Often, callbacks are used to enable the unsuitable languages to process data that comes in as streams. However, in most cases this leads to the so-called callback hell [72, 205, 208]. These problems are solved with languages and paradigms that are designed for applications that have to process and compute streams of data. Therefore, to facilitate situation-aware embedded edge computing, two concepts are proposed. First, for general purpose computations in embedded edge computing a reactive language is presented.

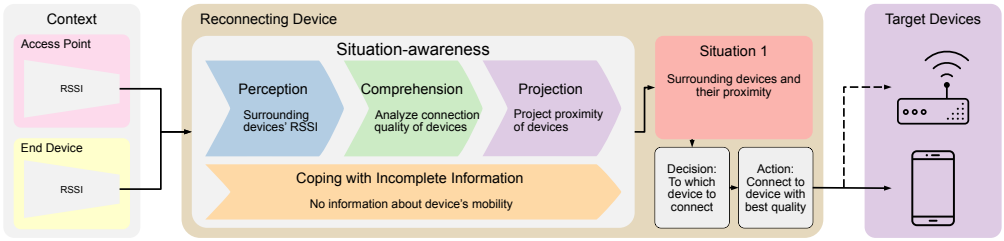


Figure 6.1: Situation-aware embedded edge computing

It supports perception, comprehension, and projection using combinators, such as *map* and *fold*, and domain-specific actions, such as transmitting aggregated data to other devices. Second, a CEP-based approach is presented. It follows the CEP paradigm, supports perception of information streams, and comprehension and projection using pre-defined aggregators. Decisions are made based on thresholds that in turn trigger domain-specific actions.

Figure 6.1 shows a novel situation-aware application in which an embedded device perceives the signal strength of surrounding devices and comprehends its situation with respect to the connection quality of the neighboring devices. Based on the projection of the proximity to the respective neighboring devices, a decision is then made about a device to establish a wireless connection to. Using this approach, embedded devices will always connect to the device with the highest signal strength, ensuring the best possible connection quality.

6.2 ReactiFi: Reactive Programming of Wi-Fi Firmware on Mobile Devices

This section presents ReactiFi, a reactive language dedicated to programming embedded systems found in embedded edge computing environments, but also in devices such as smartphones, e.g., their Wi-Fi chips. It is argued that embedded devices should be programmed at a high level of abstraction without expert knowledge and in a platform independent way. Both embedded devices and situation-aware applications are quite diverse and have to adapt to quickly changing requirements. ReactiFi is conceptually similar to functional reactive programming languages, e.g., FrTime [63] and REScala [207]. Like these languages, it provides programming abstractions for defining computations that are automatically triggered on data arrival and can be composed using functional combinators. ReactiFi differs from other reactive languages in features that address specific needs of embedded devices. The ReactiFi compiler handles platform-specific compilation and bindings to platform-specific APIs. As a result, it is possible to program embedded devices in a platform-independent manner at a high-level of abstraction. ReactiFi offers only fixed-size types, which together with its event-based dataflow programming model enables static reasoning about memory usage of applications. Furthermore, ReactiFi limits dynamic changes of dependencies between computations to dynamic branching, enabling static reasoning about the order of computations. A case study is presented to demonstrate the benefits of ReactiFi in realistic applications in a novel functionality: an adaptive file sharing application by dynamically switching to the most suitable Wi-Fi communication mode. For this purpose, the signal strength of surrounding devices is perceived on the basis of which the situation regarding the connection quality of the neighboring devices is comprehended. The projection of the connection quality then serves as a basis for deciding to which neighboring device a connection should be established. By executing this situation-aware applications on the Wi-Fi chip of a smartphone, it is shown that power consumption can be reduced by up to $87\,\%$, and that data throughput is increased by a factor of up to 3.3. The case study also demonstrates the advantages of using a high-level dataflow language for situation-aware embedded edge computing: the ReactiFi program for adaptive file sharing is 9x shorter than a corresponding low-level C program. Moreover, it is platform independent and has a clear design structure with explicit dataflow paths that are easy to follow and reason about. Finally, the benefits of ReactiFi abstractions in terms of code complexity and platform independence come without regrets: the empirical evaluation shows that there is no runtime performance overhead compared to manually written C code.

Subsection 6.2.1 introduces the design of the ReactiFi language, while Subsection 6.2.2 presents the semantics and the type system of ReactiFi. The compiler is shown in Subsection 6.2.3 followed by a discussion of guarantees ReactiFi makes in Subsection 6.2.4. Subsection 6.2.5 evaluates ReactiFi. Finally, Subsection 6.2.6 discusses related work.

Parts of this section have been published previously [235].

6.2.1 ReactiFi by Example

We introduce ReactiFi's concepts by discussing implementations of a case study that is also used in the evaluations in Section 6.2.5. It illustrates functionality that relies on information not always available in the kernel or in user space.

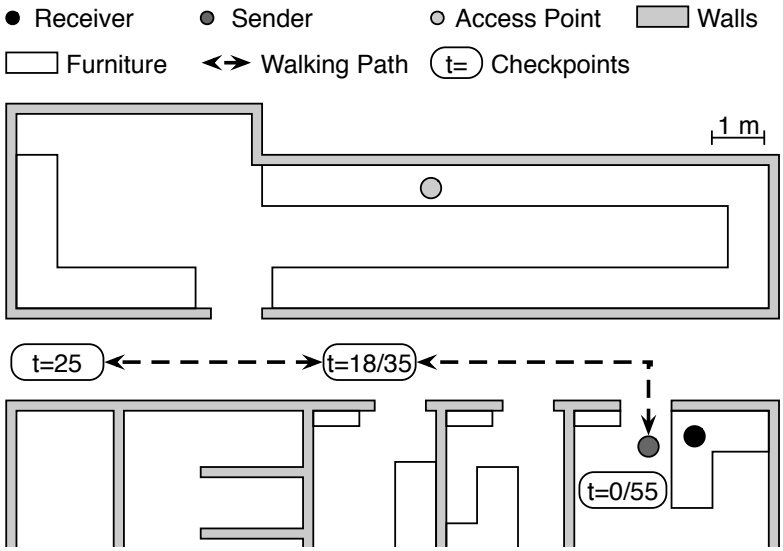


Figure 6.2: Scenario for the adaptive file sharing case study. The sender walks back and forth starting (t=0) and ending (t=55) near the receiver

Adaptive File Sharing In this case study, we use ReactiFi to implement file sharing directly on the Wi-Fi chip. File sharing in local wireless networks is a common service, e.g., Apple "AirDrop", Microsoft "Nearby Sharing", and Google "Nearby" are APIs and functionalities for such applications. Unlike these solutions, however, our file sharing application automatically switches from a direct connection between Wi-Fi devices (IEEE 802.11z Tunneled Direct Link Setup (TDLS)) to routing over the nearest access point (IEEE 802.11n AP mode), if that connection is better. This improves throughput and reduces Wi-Fi congestion.

Figure 6.2 depicts our scenario. When a file is distributed in a local wireless network, the sender as the source of the file transmits the data to an AP that relays it to the receiving destination of the file, resulting in two data streams with the same payload. If the receiver detects that the sender is in proximity (as it is the case in the beginning and at the end of the scenario timeline in Figure 6.2), it switches from AP to TDLS to establish a direct communication tunnel to the Wi-Fi device, without losing or disturbing the previously established connection to the AP. This kind of application has to be implemented on the Wi-Fi chip, since the required information is not accessible in the operating system, and the kernel cannot switch from 802.11n to 802.11z.

A ReactiFi program consists of reactive definitions – called *reactives* – that encode individual processing steps triggered by incoming events. In listing 7.1, all bold keywords (except *val*) define reactives for operations such as filters, transformations, and aggregations. Reactives may be parameterized with functions, e.g., to specify which values are filtered by a *filter* reactive. Function bodies (in braces) are written using C code embedded in ReactiFi– they operate on simple values, may access only their parameters (but not reactive definitions), and influence the dataflow only via return values. Reactives can be given names (*val*) and they can be composed (via the "." notation) into an acyclic dataflow graph.

Listing 6.1 shows the ReactiFi code. On a high level, the application computes the signal-to-noise ratio (SNR) (Line 15) for other devices and each supported mode (direct or via AP).

The SNRs are stored in a hashmap based on a key derived from the source address of the frame and whether it is a direct frame or a routed frame. To compute the keys, we first use *filter* reactives (Lines 7 and 10) to separate the frames into the two types, then use *map* reactives to generate keys from each type of frame. The two types of keys are combined using the *choice* reactive (——) in Line 12. *Choice* reactives propagate the value whenever either the left or right reactive triggers, using the left input if both trigger. Then, the SNRs for the source address of the current frame are fetched from the hashmap (Line 24) and, depending on which mode we are in (*c1* or *c2*), we compute a boolean on whether TDLS should be enabled, which is then sent to the ReactiFi runtime to ensure correct execution (line 28).

```
1  def src_key = frame ⇒ { compound_key(frame.src, FROM_SRC) }
2  def ap_key = frame ⇒ { compound_key(frame.src, FROM_AP) }
3  val monitor = Source(Monitor)
4  val frames = monitor.filter(frame ⇒ { frame.dst == ADDR })
5  val count = frames.fold({ 0 })((count, frame) ⇒ { count + 1 })
6
7  val fromSource = frames.filter(frame ⇒ {
8    frame.type == FROM_SRC_TO_AP || frame.type == FROM_SRC_TO_DST
9  })
10 val fromAP = frames.filter(frame ⇒ { frame.type == FROM_AP_TP_DST })
11
12 val keys = fromSource.map(src_key) || fromAP.map(ap_key)
13 val foreign_keys = fromSource.map(ap_key) || fromAP.map(src_key)
14
15 val avgSnrPerSrc = (count, frames, keys)
16   .fold({ hashmap_new() })((acc, count, frame, key) ⇒ {
17     int avg = hashmap_get(acc, key);
18     if (avg == MAP_ENTRY_MISSING) {
19       return hashmap_put(acc, key, frame.snr);
20     } else {
21       return hashmap_put(acc, key, avg + (frame.snr - avg) / count);
22     }
23   })
24 val c1 = (avgSnrPerSrc, fromSource, keys, foreign_keys)
25   .map((avgs, frame, k, fk) ⇒ { hashmap_get(avgs, k) > hashmap_get(avgs,
       fk) })
26 val c2 = (avgSnrPerSrc, fromAP, keys, foreign_keys)
27   .map((avgs, frame, k, fk) ⇒ { hashmap_get(avgs, k) < hashmap_get(avgs,
       fk)})
28 (c1 || c2).observe(SetTDLS)
```

Listing 6.1: ReactiFi program for adaptive file sharing

Dataflow Graph and Event/Data Propagation A ReactiFi program is transformed into a dataflow graph (DG) (cf. Section 6.2.3) that represents the abstract program logic. Each reactive r is a node in the DG with incoming edges from all inputs of r. Reactives must be declared before they are used, thus the DG is always acyclic. The DG guides the process of handling incoming events. A source is automatically triggered on arrival of incoming events from the firmware. The reactions are transitively propagated along DG paths, during which derived reactives transform, filter, and aggregate the results of other reactives, or the state in *fold* reactives gets updated. At the end of the propagation process, external effects of triggered observers are executed in the Wi-Fi firmware. For illustration, the DGs of our case study is shown in Figure 6.3 (the meaning of the blue boxes is explained in Section 6.2.3).

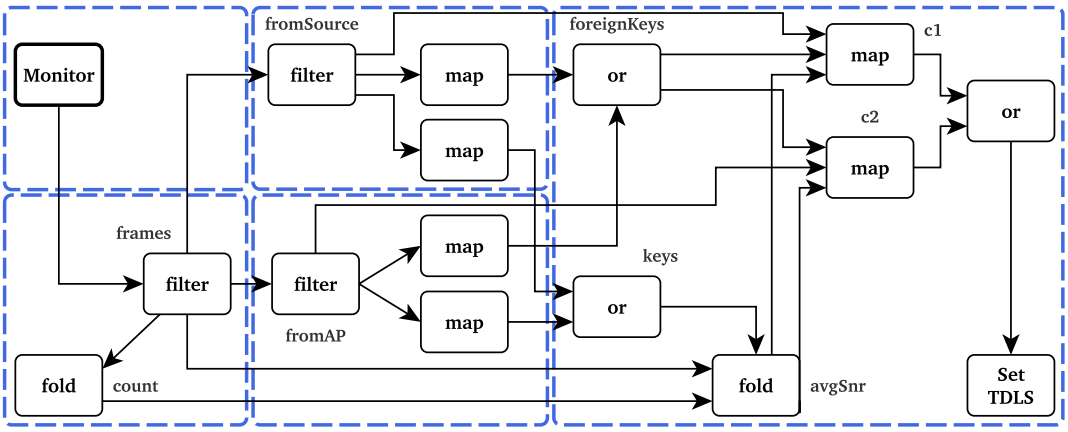


Figure 6.3: Dataflow graph of the adaptive file sharing case study

$$
\begin{array}{lr}
x ::= \textit{identifiers} & \textit{Variables} \\
v ::= \textit{fixed size values} & \textit{Values} \\
f ::= \overline{x} \Rightarrow \{\textit{cblock}\} & \textit{Functions} \\
s ::= \texttt{Monitor} \mid \texttt{Timer}(v) \mid \ldots & \textit{Sources} \\
o ::= \texttt{SendToOs} \mid \texttt{SwitchChannel} \mid \ldots & \textit{Effects} \\
r ::= \texttt{Source}(s) \mid \overline{x}.\texttt{map}(f) \mid \overline{x}.\texttt{fold}(v)(f) \mid \texttt{fold}(v)(\overline{x \rightarrow f}) \mid & \textit{Reactives} \\
\quad (x||x) \mid x.\texttt{filter}(f) \mid x.\texttt{snapshot}(x) \mid \overline{x}.\texttt{observe}(o) & \\
d ::= \texttt{val}\ x : \mathrm{T} = r & \textit{Definitions}
\end{array}
$$

Figure 6.4: Syntax of ReactiFi

6.2.2 The ReactiFi Language: Syntax and Semantics

Figure 6.4 shows the syntax of ReactiFi. A ReactiFi program is a sequence of definitions d of the kind $\texttt{val}\ x = r$, each denoting a reactive expression r by an identifier x. A reactive r is either a source without inputs, or is derived from its input reactives $\overline{x}$ (shorthand for $(x_1, \ldots, x_n)$) using one of the combinators *map*, *fold*, ——, *filter*, or *snapshot*. Some combinators are parameterized by an initial value v or a function f. Values and function bodies are written in the C programming language. The examples shown in Section 6.2.1 use syntactic sugar for chained pipelines. The single assignment form used here simplifies the presentation without affecting the semantics.

To interface with the Wi-Fi chip, ReactiFi uses a set of predefined interactions. Interactions are wrapped into $(\mathit{Source}(s))$ reactives or are parameters of $(\overline{x}.\texttt{observe}(o))$ reactives. Sources include receiving frames, timers, or changed channel state. Observers include transmitting custom Wi-Fi frames, sending packets to the network stack of the host operating system, switching channels, etc. The full list of external interactions is shown in Table 6.1.

Figure 6.5 shows the typing rules of ReactiFi reactives. To simplify our presentation, we do

Table 6.1: Predefined interactions between ReactiFi application and runtime

Category	Declaration	Description
Receive	SentFrame	Outgoing frame
	ReceivedFrame	Incoming frame filtered by destination addr.
	Monitor	All incoming frames
Management	ScanResult	Results of a Wi-Fi access point scan
	ChannelState	Updated channel state information
	TxPower	Updated transmit power information
Interrupts	IOCTL	ioctl data
	Timer	Timer event with ms granularity
Effects	SendFrame	Send Wi-Fi frame
	SwitchChannel	Switch Wi-Fi channel
	ChangeCSI	Change Wi-Fi channel state information
	SetTxPower	Set transmit power
	SendToOS	Send a value frame to the operating system
	SetTDLS	Change the connection mode to/from TDLS

not show the typing context for variables. We assume each reactive may access all other reactives defined before, but not after, in the list of definitions, resulting in an acyclic graph of dependencies between reactives. All reactives in ReactiFi have the type $\mathrm{Reactive[A]}$ and are parametric over the value they carry, but are never nested. We assign semantics to individual ReactiFi reactives by giving the translation of individual definitions $\texttt{val}\ x : \mathrm{T} = r$ into C-like statements shown in Figure 6.6. Translation is written $C[\![\texttt{val}\ x : \mathrm{T} = r]\!]$). Conceptually, the translated statements will be executed in the topological order of the DG, as explained in Section 6.2.3. In general, each statement first checks if the trigger condition (e.g., $T[\![\overline{x}]\!]$) for its inputs (e.g., $\overline{x}$) are fulfilled, and then updates the current value of the reactive (x_0). If any condition for a reactive is false and there is no else branch, then the reactive itself does not trigger, stopping the propagation at this point. Transformation of functions $C[\![f(\overline{x})]\!]$ result in a call to a fresh top-level function definition. Compiling an *identifier* that binds a reactive $C[\![x]\!]$ produces code that accesses the value of that reactive.

A *source reactive* is of type $\mathrm{Reactive[A]}$, given that it is triggered by a source s of type $\mathrm{SourceDef[A]}$, i.e., the inner type A of the reactive is defined by the inner type of s (rule $\textsc{Source}$). An *observe reactive* has a single input x of type $\mathrm{Reactive[A]}$, given that they observe an o of type $\mathrm{ObserveDef[A]}$, i.e., the inner type of the input reactive must match the type that is consumed by o (rule $\textsc{Observe}$). A *filter reactive* has the same type, $\mathrm{Reactive[A]}$, as its input x. To use the function f for filtering, the function must take a parameter of the type A and return a $\mathrm{Boolean}$ (rule $\textsc{Filter}$). *Filter* reactives pass the value of their input unchanged, if the *filter* condition $f(x)$ is true. A *choice reactive* (——) takes two inputs x_1 and x_2 which must be of the same type, and the result is also of that same type (rule $\textsc{Choice}$). It returns the value of its left operand x_1 if x_1 triggers, otherwise the right operand x_2 if x_2 triggers. The type of *map reactive* says that the i-th input reactive x_i must match the type of the i-th parameter A_i of the given function f. The resulting type is $\mathrm{Reactive[R]}$ where R is the result type of f. For example, a *map* reactive that combines three reactives with types $\mathrm{A, B, C}$ as inputs expects a function of type $(\mathrm{A, B, C}) \Rightarrow \mathrm{R}$. The value when a map reactive r triggers is function f applied to the value of all inputs $\overline{x}$ of r.

$$\textsc{Source}\ \frac{s : \text{SourceDef[A]}}{\texttt{Source}(s) : \text{Reactive[A]}} \qquad \textsc{Observe}\ \frac{x : \text{Reactive[A]} \qquad o : \text{ObserverDef[A]}}{x.\texttt{observe}(o) : \text{Observer}} \qquad \textsc{Subtype}\ \frac{x : \text{Fold[A]}}{x : \text{Reactive[A]}}$$

$$\textsc{Filter}\ \frac{x : \text{Reactive[A]} \qquad f : \text{A} \Rightarrow \text{Boolean}}{x.\texttt{filter}(f) : \text{Reactive[A]}} \qquad \textsc{Choice}\ \frac{x_1 : \text{Reactive[A]} \qquad x_2 : \text{Reactive[A]}}{(x_1 || x_2) : \text{Reactive[A]}}$$

$$\textsc{Map}\ \frac{x_i : \text{Reactive[A}_\text{i}\text{]} \qquad f : \overline{\text{A}} \Rightarrow \text{R}}{\overline{x}.\texttt{map}(f) : \text{Reactive[R]}} \qquad \textsc{Fold}\ \frac{x_i : \text{Reactive[A}_\text{i}\text{]} \qquad v : \text{R} \qquad f : (\text{R}, \overline{\text{A}}) \Rightarrow \text{R}}{\overline{x}.\texttt{fold}(v)(f) : \text{Fold[R]}}$$

$$\textsc{FoldAll}\ \frac{x_i : \text{Reactive[A}_\text{i}\text{]} \qquad v : \text{R} \qquad f_i : (\text{R}, \text{A}_\text{i}) \Rightarrow \text{R}}{\texttt{fold}(v)(\overline{x \rightarrow f}) : \text{Fold[R]}} \qquad \textsc{Snapshot}\ \frac{x_1 : \text{Reactive[A]} \qquad x_2 : \text{Fold[B]}}{x_1.\texttt{snapshot}(x_2) : \text{Reactive[B]}}$$

Figure 6.5: Typing rules of reactives in ReactiFi

$$\begin{aligned}
C[\![\texttt{val } x_0 = \texttt{Source}(s)]\!] &= \texttt{if } (T[\![s]\!])\{x_0 = C[\![s]\!]\} \\
C[\![\texttt{val } x_0 = \overline{x}.\texttt{map}(f)]\!] &= \texttt{if } (T[\![\overline{x}]\!])\{x_0 = C[\![f(\overline{x})]\!]\} \\
C[\![\texttt{val } x_0 = \overline{x}.\texttt{fold}(v)(f)]\!] &= \texttt{if } (T[\![\overline{x}]\!])\{x_0 = C[\![f(x_0, \overline{x})]\!]\} \\
C[\![\texttt{val } x_0 = \texttt{fold}(v)(x \rightarrow f)]\!] &= \texttt{if } (T[\![x]\!])\{x_0 = C[\![f(x_0, x)]\!]\} \\
C[\![\texttt{val } x_0 = \texttt{fold}(v)(x_1 \rightarrow f_1, \overline{x \rightarrow f})]\!] &= \texttt{if } (T[\![x_1]\!])\{x_0 = C[\![f_1(x_0, x_1)]\!]\}; \\
&\qquad C[\![\texttt{val } x_0 = \texttt{fold}(v)(\overline{x \rightarrow f})]\!] \\
C[\![\texttt{val } x_0 = (x_1 || x_2)]\!] &= \texttt{if } (T[\![x_1]\!])\{x_0 = C[\![x_1]\!]\} \\
&\qquad \texttt{else if } (T[\![x_2]\!])\{x_0 = C[\![x_2]\!]\} \\
C[\![\texttt{val } x_0 = x.\texttt{filter}(f)]\!] &= \texttt{if } (T[\![x]\!] \&\& C[\![f(x)]\!])\{x_0 = C[\![x]\!]\} \\
C[\![\texttt{val } x_0 = x_1.\texttt{snapshot}(x_2)]\!] &= \texttt{if } (T[\![x_1]\!])\{x_0 = C[\![x_2]\!]\} \\
C[\![\texttt{val } x_0 = C[\![\overline{x}.\texttt{observe}(o)]\!]]\!] &= \texttt{if } (T[\![\overline{x}]\!])\{C[\![o(\overline{x})]\!]\}
\end{aligned}$$

Figure 6.6: Compiling individual ReactiFi reactives (left) to C code (right)

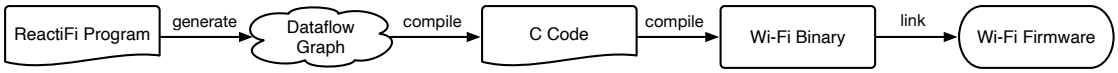


Figure 6.7: Stages of compiling ReactiFi to Wi-Fi chips

A *fold reactive* has type $\mathrm{Fold}[\mathrm{R}]$ which is a subtype of $\mathrm{Reactive}[\mathrm{R}]$ (rule $\textsc{Subtype}$). The $\mathrm{Fold}[\mathrm{R}]$ type identifies stateful reactives. This type distinction is used to limit which reactives are accessed by *snapshot* reactives. There are two syntactic forms for *fold* reactives. Both are of type $\mathrm{Fold}[\mathrm{R}]$ with an initial value v of type R. The first syntactic form, $\overline{x}.\texttt{fold}(v)(f)$, has inputs and a function with matching types, similar to *map* reactives. In addition to *map* reactives, *fold* reactives have their current value x_0 stored globally. If all inputs $\overline{x}$ are triggered, then x_0 is updated by applying f to x_0 and all inputs $\overline{x}$. The second syntactic form, $\texttt{fold}(v)(\overline{x \to f})$, called *fold all* reactives, has one function per input. *Fold all* reactives are translated to multiple statements, each statement updates the current state x_0 by applying $x_0 = f_i(x_0, x_i)$, if the corresponding input x_i is triggered, otherwise the application of that f_i is skipped. All f_i with triggered x_i are applied in the order they are defined, potentially updating x_0 multiple times. The typing rule $\textsc{FoldAll}$ ensures that the second parameter A_i of each f_i matches with the type of the assigned input x_i.

A *snapshot reactive* takes two inputs, where the first x_1 is a reactive of type $\mathrm{Reactive}[\mathrm{A}]$ and the second x_2 is a *fold* of type $\mathrm{Fold}[\mathrm{B}]$ and results in a $\mathrm{Reactive}[\mathrm{B}]$ (rule $\textsc{Snapshot}$). A *snapshot* reactive triggers when its first input x_1 triggers, but returns the value of its second input x_2. The type restriction on x_2 is because only *fold* reactives have a defined state when they do not trigger.

6.2.3 The ReactiFi Implementation

A ReactiFi program is processed in four steps (Figure 6.7): (i) the dataflow graph (DG) is constructed, (ii) a C program is generated from the DG, (iii) the C source code is compiled into a binary, which (iv) is loaded to the Wi-Fi chip and linked into the firmware at runtime.

Generating and Typechecking the Dataflow Graph

ReactiFi is implemented as an embedded domain-specific language (EDSL) in Scala, i.e., its abstractions are implemented as a Scala library. We selected Scala due to its support for embedding DSLs, e.g., we implement the DSL to reuse Scala's type checker for the typing rules in Figure 6.5. A ReactiFi program consists of a set of library calls that look like proper DSL syntax. These calls construct the DG that is subsequently compiled to C. User-defined functions are opaque to the Scala DSL – they are directly copied into the generated C code. Once constructed, the DG is analyzed to extract the following information that is passed to the subsequent processing phases: (i) a topological order of all reactives, (ii) a set of conditions guarding the activation of each reactive, and (iii) types and memory requirements of reactives in the DG.

Generating C Code for the Update Function

Given the DG and the embedded C code, the ReactiFi compiler generates a single sequential update function (UF) in C, which implements the reaction to external events. Thereby, the compiler performs the following domain-specific optimizations:

Static Sequential Scheduling The DG specifies a logically concurrent execution order of reactives in response to individual external events; moreover, only a subset of reactives is typically triggered for each external event. Wi-Fi chips only support sequential execution and network applications are often latency-sensitive. To address these constraints, the compiler (i) sequentializes the order of updating reactives and (ii) generates a minimum number of conditional branches to select the updated reactives. While the relative execution order of reactives can be statically fixed according to the topological order of them in DG, whether and when reactives trigger depends on runtime conditions. Sources and *filter* reactives define new conditions. For all other reactives, the conditions are derived from the conditions of their input reactives. The *choice* and *fold all* reactives use the disjunction of the conditions of all inputs. All other reactives use the conjunction of the conditions of all inputs. Reactives in the DG are grouped into uninterrupted pipelines based on shared filtering conditions. The compiled update function only checks conditions once per group. For illustration, consider the DG of the file sharing case study in Figure 6.3; the blue boxes mark the uninterrupted groups; for instance, the rightmost group will execute only if the *Monitor* source fires, the condition for the *frames* filter holds, and either *fromSource* or *fromAP* are true.

Optimized Memory Management Wi-Fi chips have limited memory. For instance, the memory built into the Nexus 5 used in our evaluation has $768\,\text{kB}$ RAM, most of which is used by the basic firmware, with only as little as $100\,\text{kB}$ RAM left for higher-level functionality; to put this into perspective, a single IP packet is up to $2\,\text{kB}$ in size. Reactives are abstractions with zero runtime memory cost, i.e., $\text{sizeof}(\text{Reactive}[\text{T}]) == \text{sizeof}(\text{T})$. To facilitate compile-time estimation of the needed memory, ReactiFi allows only fixed size types T to be used in code. Memory for reactives is reclaimed at the earliest time possible. Technically, to find the reclamation point of a reactive r_1, the compiler traverses the sequential execution order from the back to find the last reactive r_2 that depends on r_1. The scope of r_1 extends from r_1 until after r_2. Unlike other reactives, the state of *fold*s is stored between updates, thus never reclaimed. Overall, for each reactive r, the compiler knows how much memory is already allocated when a new value will be computed for r. This is the sum of the memory allocated to all *fold*s in the program plus the sum of the memory allocated to all non-fold reactives in scope. This way, the compiler is able to maximize the memory available for executing the function bodies embedded in the reactives.

Exemplary Compilation For illustration, consider the code below, defining a *map* reactive (*address*) with two inputs, *frame* and *subframe*; we assume that both frames are derived from a *Monitor* source (not shown for brevity).

```
1 val address = (frame, subframe).map((fr, sub) ⇒ {/* extractAddress */})
```

The generated C code is sketched in Listing 6.2. Since there are no folds, the program has only local state. The user-defined function is extracted to a top-level C function `extractAddress`. Within the `update` function, first, the variables for the source conditions (`monitor_condition`), the values computed for the input reactives (`frame_value`, `subframe_value`), and for the *map* reactive (`address_value`) are declared. Then, the trigger conditions of sources are computed, followed by the guarded execution of reactives, when the sources are triggered. The code illustrates the two compiler optimizations. First, the trigger condition is only checked once for all reactives as opposed to once per reactive. Second, the values *frame* and *subframe* are deallocated immediately after they are used and no longer needed. We assume that `address_value` is used later in the program, otherwise the whole program would be optimized away.

```
1  address_t extractAddress(frame_t fr, frame_t sub) { /* extractAddress */
       }
2  // state of fold reactives would be above
3  void update() {
4    bool monitor_condition;
5    frame_t frame_value;
6    frame_t subframe_value;
7    address_t address_value;
8
9    monitor_condition = runtime_is_triggered(Monitor)
10   if (monitor_condition) {
11     frame_value = ...;
12     subframe_value = ...;
13     address_value = extractAddress(frame_value, subframe_value);
14     deallocate(frame_value);
15     deallocate(subframe_value);
16   }
17 }
```

Listing 6.2: Generated C code example

From C to Binary to the Wi-Fi Chip

There are several available deployment targets. Some of them, e.g., SoftMAC Wi-Fi dongles [215], or Espressif's ESP platform [164], are special-purpose hardware with custom software. On the contrary, the Nexmon firmware patching framework [213, 214] can also be used and executed on off-the-shelf smartphones. Since we want to deploy ReactiFi programs on off-the-shelf smartphones for validating the feasibility of our approach in a real-world case study, our current implementation uses Nexmon as the target platform. To transfer the compiled ReactiFi program to the Wi-Fi chip, we created a new `ioctl`. Such `ioctl`s are common communication channels between the host (either kernel or user space application) and dedicated hardware components like the Wi-Fi chip. At this point, the dynamic linker usually performs a relocation step to adjust the addresses of branches to their absolute memory location of the loaded code. This relocation step, however, would require the Wi-Fi chip to be restarted, making Wi-Fi communication temporarily unavailable. However, we want to reconfigure the Wi-Fi chip at runtime without disturbing ongoing connections. Therefore, we extended Nexmon to support *Position Independent Code* (PIC). PIC modules can be loaded to arbitrary memory addresses from where their execution can be triggered during runtime. Since the PIC module is unaware of where the binary blob gets loaded, the code

performs jumps relative to the program counter. Existing firmware functions, on the other hand, are accessed by first loading the absolute target address from the *Global Offset Table* to a register and then jumping to that address. To recap, our PIC extensions enable loading and executing ReactiFi programs without restarting the Wi-Fi chip. This allows ReactiFi to always ensure basic functionality of the IEEE 802.11 specification.

6.2.4 Advantages of Using ReactiFi for Wi-Fi Programming Compared to C

In this section, we summarize how ReactiFi's compiler and runtime address domain-specific issues of the Wi-Fi platform. In addition, to help the reader appreciate the benefits of using a high-level dataflow language in terms of code quality, we walk through a C implementation of the adaptive file sharing application shown in the Appendix, and compare the latter with the ReactiFi implementation shown in Section 6.2.1.

Properties Ensured by the ReactiFi Compiler and Runtime

ReactiFi's declarative dataflow programming model enables the compiler to ensure several properties, as described below.

Minimized Memory Usage ReactiFi's programming model matches well the assumption that Wi-Fi chips are supposed to quickly react to incoming events, but only store limited state. First, data pertaining to an event only exists for the duration of the event. Unused or inactive reactives and their derived reactives are not executed or initialized at all and, hence, do not consume any memory. Second, except for *fold*s, other reactives do not require to store state between updates. Third, the ReactiFi compiler ensures that temporarily used memory is freed as soon as possible during updates (cf., Section 6.2.3). Finally, usage of memory by reactives is statically limited in size. ReactiFi only supports parameters to reactive computations with a statically bound size. The usage of any other types is prohibited by the type checker.

Automatic, Correct, and Optimized Scheduling The DG allows precise and sound reasoning about the order of reactive computations, enabling compile-time optimizations and scheduling without any runtime overhead. This is possible because ReactiFi limits dynamic changes of the DG and the scheduling order to filtering. It has been argued that the limited expressiveness is sufficient for most programs [257]. As a result, the ReactiFi compiler is free to rearrange the order of execution as long as explicit dependencies between reactives are preserved, allowing to minimize dynamic checks (cf., Section 6.2.3).

Platform Independence and Compliance ReactiFi is compliant with the IEEE 802.11 specification by always providing basic functionality of the Wi-Fi firmware. ReactiFi programs cannot break basic functionality of the Wi-Fi firmware. Interactions only happen through high-level source reactives and observers (cf., Section 6.2.1 and Table 6.1). Furthermore, the generated code can be deployed on a Wi-Fi chip without interruption (cf., Section 6.2.3). Thus, ReactiFi allows developers with no particular Wi-Fi expertise to write platform-independent Wi-Fi functionality, leaving error-prone and platform-specific aspects to be handled by ReactiFi.

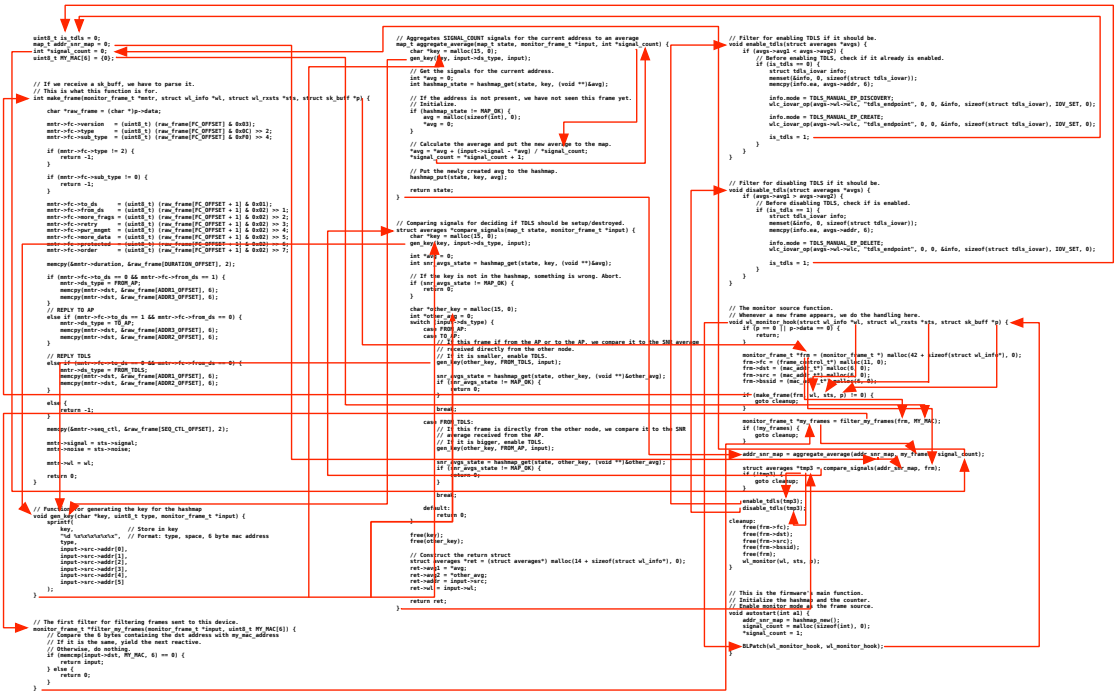


Figure 6.8: Dataflow of the C implementation of the adaptive file sharing case study; The arrows represent the direction of the dataflow

Non-Ensured Properties ReactiFi cannot reason about user-defined C code encapsulated in reactives. However, the amount of C code necessary can be kept to a bare minimum, and it is sufficient to review each function individually. While each C function is typechecked by the C compiler, and ReactiFi ensures type correctness of using the function in the DG, ReactiFi cannot ensure that C functions terminate or use a bounded amount of memory. Beyond these type checks and the enforcement of the fixed-sized types, the current type checker of ReactiFi only inherits the standard guarantees of the Scala type checker. In principle, it is possible to extend the type system with user-defined specifications about the behavior of C blocks, e.g., with regard to real-time behavior. With such specifications and the explicit knowledge about the DG, the compiler can reason about real-time guarantees of ReactiFi programs. Such extensions remain to be investigated in future work.

Comparison to C

We compare ReactiFi to C, because C is the most widely used language for programming Wi-Fi chips. Yet, our arguments apply to all imperative languages that do not support a declarative dataflow programming model[1].

To start with, consider how much code complexity and programming effort ReactiFi saves. While the ReactiFi program is only 28 LoCs long (Listing 6.1), the C program Listing 6.5

[1]The adaptive file sharing application was implemented in C by me. I have ample experience with C programming and the Nexmon platform. While I am aware that this is a threat to the validity of the statements I make in the following, there are no existing programs for the Wi-Fi chip, which I could have used – ultimately, the starting point is that Wi-Fi chips are not programmable today.

in Section 6.2.8 (page 139) even without `#include` directives, comments, and empty lines consists of 229 LoCs (almost 9× more!). Moreover, the dataflow of the C implementation shown in Figure 6.8 is over-proportionally complex and hard to reason about with data dependencies modeled implicitly using global state, side effects, and pointers. In contrast, the dataflow of the ReactiFi implementation of the same functionality shown in Figure 6.3 is simple and explicit in the program.

Moreover, a programmer that implements Wi-Fi applications in C needs detailed low-level knowledge of the specific platform. Consider as an example the extract from the C implementation of the adaptive file sharing case study in Listing 6.3. To enable monitor mode, the developer must know the memory address of the *Monitor* source for every target Wi-Fi chip and firmware version. Similarly, the developer must often use low-level bitwise operations to implement data extraction from Wi-Fi frames. This requires knowledge of the internal data structures, since information might be stored in different locations inside the frame, depending on the conditions, e.g., the source and destination addresses of a frame must be extracted differently, depending on whether the frame originates from an access point or not. As an example, Listing 6.4 shows how to get the type field of a Wi-Fi frame.

```
1 __attribute__((at(0x18DA30, "", CHIP_VER_BCM4339,
      FW_VER_6_37_32_RC23_34_40_r581243)))
2 __attribute__((at(0x18DB20, "", CHIP_VER_BCM4339,
      FW_VER_6_37_32_RC23_34_43_r639704)))
3 BLPatch(wl_monitor_hook, wl_monitor_hook);
```

Listing 6.3: Enabling monitor mode using C

```
1 mntr->fc->type      = (uint8_t) (raw_frame[FC_OFFSET] & 0x0C) >> 2;
2 mntr->fc->sub_type  = (uint8_t) (raw_frame[FC_OFFSET] & 0xF0) >> 4;
```

Listing 6.4: Parsing type fields of a Wi-Fi frame in C

The above observations indicate that ensuring correctness is difficult in an imperative language like C. It requires highly skilled programmers to understand the program by manually tracking memory allocation along the complex dataflow graph. This makes it very hard for the programmer to assess whether the memory needs of the application can be satisfied by the available memory. Furthermore, incorrect execution order may lead either to memory corruption or inefficient execution. Ensuring that this cannot happen is cumbersome and error-prone: it is necessary to repeatedly test the program to ensure correct order of executions; even providing basic operation requires enormous effort. What is more, the efforts will have to be repeated over and over for any new program, as there is no automated language machinery available. Of course, it is theoretically possible in C to provide higher-level APIs that hide some low-level details of the platform. However, such libraries would only partly solve the outlined problems, since providing the memory, scheduling, and platform independence of ReactiFi fundamentally requires to reason about the DG. Furthermore, it is unclear how to ensure that APIs are correctly used [11] and this is especially problematic for APIs abstracting a diverse set of low-level platform-dependent details. At the end, given that ReactiFi's dataflow abstractions come without runtime overhead (as we show in the next section), no good arguments are left for an API-based approach.

6.2.5 Empirical Evaluation

First, we quantify the basic power consumption and performance of the Wi-Fi chip using a micro-benchmark. Next, we experiment with our case study to validate our claims regarding improved power consumption and throughput.

Experiments Using a Micro-benchmark

The Benchmark Since currently there are no other programming languages, implementations, or applications for Wi-Fi chips in off-the-shelf smartphones, we cannot perform comparative benchmarks using ready to use applications. Therefore, we adapted parts of the Linux Internet Control Message Protocol (ICMP) implementation. ICMP is used by nodes in the network to send control messages like indicating success communicating with other nodes. Since the Linux implementation is deeply embedded into the operating system kernel, it is nearly impossible to extract the entire code. Instead, we adapted the ICMP code for handling ICMP *echo* packets, as commonly found in the "ping" utility. Using our ICMP adaptation, we can measure basic power consumption and latency executed in three different environments, i.e., user space, operating system kernel, and Wi-Fi chip.

Experimental Setup We placed two Nexus 5 smartphones about $30\,\text{cm}$ apart from each other. The first Nexus 5 sent ICMP *echo-requests*, while the second one processed the received frames using two different versions of our ICMP program, both returning an *echo-reply* at the end of the execution. The first version is implemented in pure C where ReactiFi was not involved and the ICMP *echo-requests* are handled in the same way as in the Linux kernel, sending an ICMP *echo-reply* without further computations. The second version of our ICMP program, written in ReactiFi, contains single *filter* (dropping non-ICMP *echo-requests*), *map* (computing the ratio between Wi-Fi frame size and ICMP packet size), and *fold* (counting the received ICMP *echo-requests*) reactives, and returns a corresponding *echo-reply* after the respective reactives. Each of the reactives were used in separate tests. The pure C version of the program enables us to compare the power consumption and latency to the ReactiFi implementation. During the experiments, MAC Protocol Data Unit Aggregation (A-MPDU) and frame re-transmission were disabled in the Wi-Fi firmware. Thus, every packet was sent once in a separate Wi-Fi frame, allowing an evaluation for each packet. To evaluate latency, we measured the ICMP round trip time for each packet. For a high-resolution time measurements without interference, an external device was used to capture ICMP packets between the two Nexus 5 smartphones using Wireshark[2]. To measure power consumption without interference from the battery, we removed the battery from the Nexus 5 and the charge controller from the battery, soldered wires to the charge controller, and put the controller without the battery back into the Nexus 5. The measurements were performed using a Monsoon High Voltage Power Monitor with a sample rate of $5\,\text{kHz}$ and a resolution of $286\,\mu\text{A}$. The voltage was set to $4.2\,\text{V}$, which corresponds to about $92\,\%$ battery capacity.

[2] `https://www.wireshark.org`, last accessed 2020-10-01.

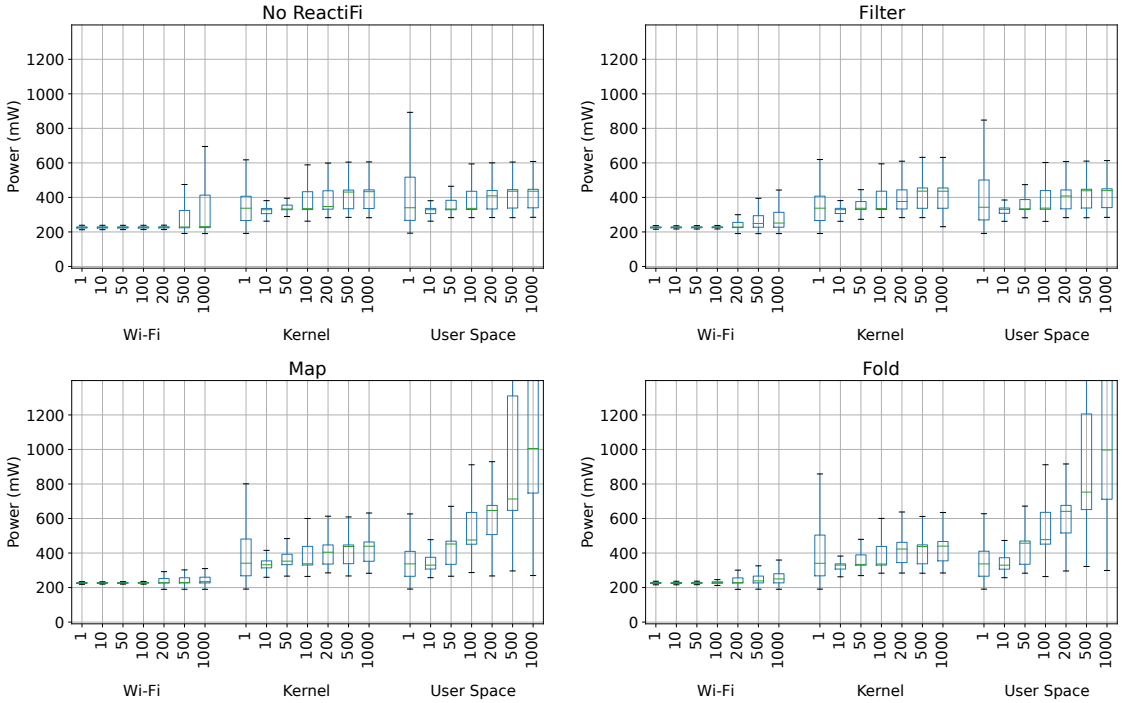


Figure 6.9: Power consumption: Wi-Fi, OS kernel, and user space

Power Consumption Figure 6.9 shows the power consumption of the micro-benchmark. Each subplot illustrates a different test case, the x-axis shows the number of requests per second, grouped by execution environment. The y-axis denotes the power consumption. Note that for user space tests with high data rates, the boxes where truncated so that the Wi-Fi based tests can be seen better.

The ICMP program executed on the Wi-Fi chip saves up to $73\,\%$ power compared to the user space implementation and $30\,\%$ compared to the in-kernel execution, regardless of which program version was used, since the main CPU, which consumes more power in general, has to process all frames. Given that for low packet rates the kernel falls into power saving mode and user space applications are suspended and both need to be woken up quite often, tests with 1 request per second need more power than tests with 10 requests per second. This effect disappears with higher packet rates, since the ICMP program executed in the kernel or in user space is not suspended anymore. These tests show that executing code on the Wi-Fi chip reduces the power consumption compared to execution in the kernel or in user space. Additionally, when comparing the results of the four Wi-Fi experiments shown in Figure 6.9, it is evident that reactives do not introduce significant power overhead compared to the purely C-based tests (shown in the upper left subfigure of Figure 6.9).

Latency Figure 6.10 shows the round trip times (RTT) of ICMP *echo-requests* and the corresponding *echo-replies*. The RTTs, when processing directly on the Wi-Fi chip, are always low at about $0.3\,$ms, regardless of the number of requests/s. When incorporating reactives, RTTs are slower (about $1.4\,$ms to $1.8\,$ms), though consistent. The kernel and user space tests need 5 to 7 times more, depending on the test. With low data rates, however, both the

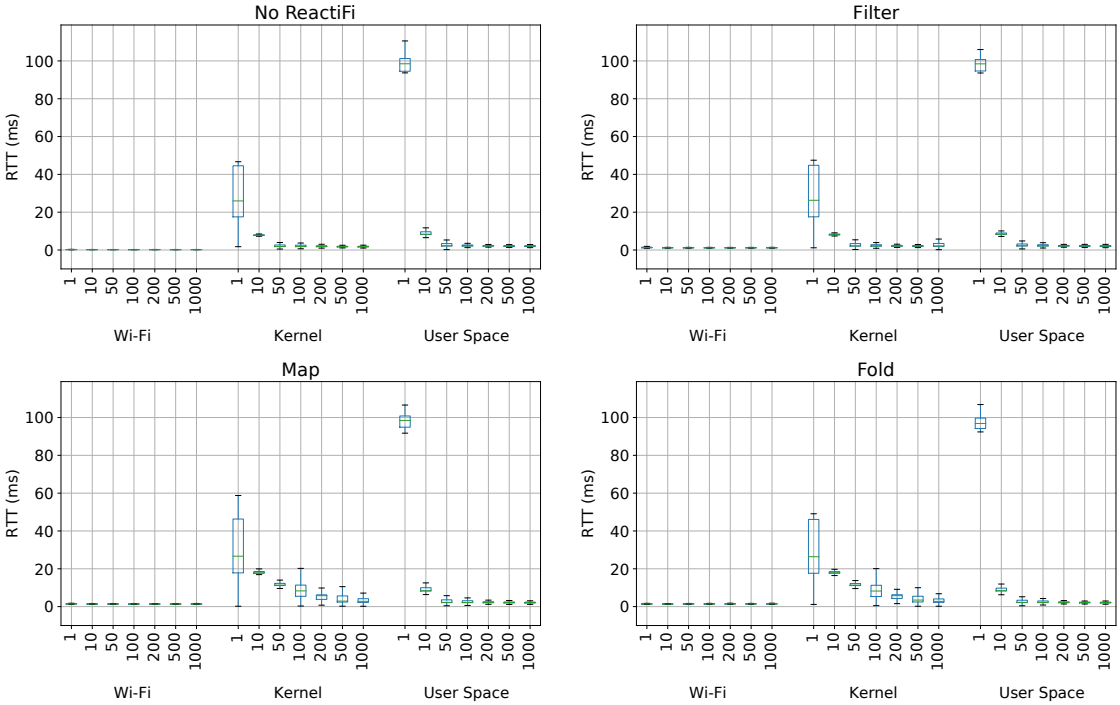


Figure 6.10: ICMP Round Trip Time: Wi-Fi, OS kernel, and user space

kernel and the user space application are suspended, resulting in significantly higher RTTs (up to about 100 ms). These tests show that executing code on the Wi-Fi chip reduces the execution time, and thus, the overall latency vs. execution in the kernel or in user space. We can further observe that the execution time using the Wi-Fi chip is more predictable across the tests. Finally, compared to the purely C-based tests (shown in the upper left subfigure of Figure 6.10), ReactiFi does not introduce significant latency overhead.

Throughput Boosting by Adaptive File Sharing

This experiment empirically validates our claim that making the Wi-Fi chip programmable enables novel networking applications with improved throughput, such as our adaptive file sharing case study.

Experimental Setup We used the ReactiFi program in Listing 6.1 on a Nexus 5 smartphone (receiver) to download a file from a Raspberry Pi 3 (sender) in the scenario shown in Figure 6.2. The file is served by a standard HTTP server without modifications. We used a Turris Omnia RTROM01 router in stock configuration as our AP. Wi-Fi was set to IEEE 802.11n mode on channel 6 in the 2.4 GHz band to increase the usable Wi-Fi range.

As illustrated in Figure 6.2, the AP was about 6 meters away from the stationary Nexus 5. In the beginning, the Raspberry Pi 3 was about 1 meter away from the Nexus 5 at $t = 0$, we then moved up to 3 m towards the AP ($t = 18$) and continued farther away from both the AP and the Nexus 5. The maximum distance between Nexus 5 and Raspberry Pi 3 was about 12 meters and 7 meters between Raspberry Pi 3 and AP ($t = 25$). After that, the same path was

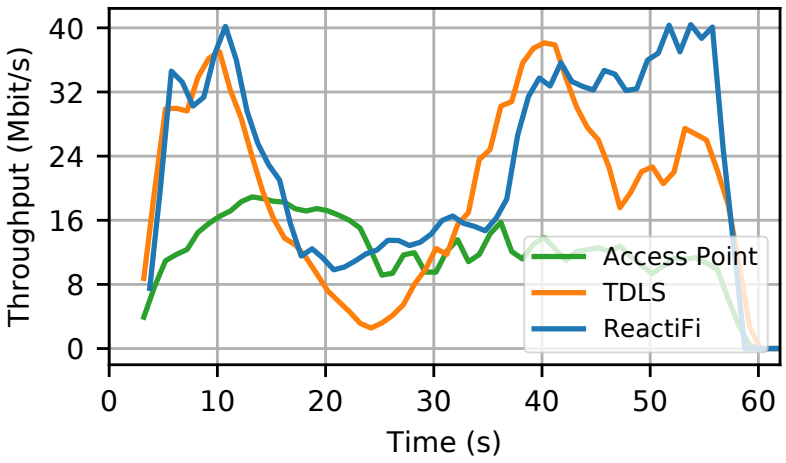


Figure 6.11: Throughput of the adaptive file sharing application

used for the way back ($t = 35$), resulting in the same position as at the beginning of the test ($t = 55$). The experiment took $55\,\mathrm{s}$ in total, while the maximum distance between Nexus 5 and Raspberry Pi 3 was reached after about $25\,\mathrm{s}$. Since all surrounding wireless traffic had to be analyzed for this application, the Nexus 5 was set to Wi-Fi monitor mode.

Throughput Figure 6.11 shows throughput in $\mathrm{Mbit\,s^{-1}}$ (y-axis) and time in seconds (x-axis) during three tests: (i) using only the AP shown in Figure 6.2 in IEEE 802.11n AP mode, (ii) using only IEEE 802.11z TDLS to establish a direct connection between the Nexus 5 and the Raspberry Pi 3, and (iii) using the ReactiFi adaptive file sharing application to automatically switch between (i) and (ii).

While the throughput of test (i) in AP mode is more or less at about $12\,\mathrm{Mbit\,s^{-1}}$ during the entire test, the same file shared via TDLS in test (ii) shows peaks at about $40\,\mathrm{Mbit\,s^{-1}}$ at the beginning and the end of the test, i.e., when the Raspberry Pi 3 is close to the Nexus 5 ($t = 0$ and $t = 55$). At the maximum distance (i.e., the worst SNR) between the devices ($t = 25$), the throughput drops to $4\,\mathrm{Mbit\,s^{-1}}$ in the TDLS-only test.

The same experiment performed with the ReactiFi program in test (iii) results in significant improvements over both the AP and TDLS tests, as shown in Figure 6.11. At the beginning and at the end (i.e., with the best SNR), the results are comparable to the TDLS-only test (ii), where throughput exceeds $40\,\mathrm{Mbit\,s^{-1}}$. However, with the worst SNR, the throughput does not fall below the values of the AP-only test (i). This experiment shows that ReactiFi enables the development of novel Wi-Fi applications that cannot be implemented in the operating system kernel or in user space and that can have significant performance gains, leading to up to a factor of 3.3 higher throughput compared to AP-only mode.

6.2.6 Related Work

Extending Network Functionality Using eBPF to program the Linux kernel network stack, multiple works propose extension systems for IPv6 [113, 260], OSPF [258], TCP [36, 113, 244], Multipath TCP [91], BGP [258], and QUIC [66]. All these approaches, however, rely on

the C language with its downsides. Additionally, these approaches cannot access information from the PHY and MAC layers.

Programmable Wi-Fi Firmware Tinnirello et al. [243] present a finite state machine for implementing MAC protocols on Wi-Fi firmware, and Bianchi et al. [31] extend this approach by introducing MAClets for simplifying the programmability of MAC protocols executed on Wi-Fi firmware. In contrast, ReactiFi supports a wide range of applications not limited to MAC protocols executed on off-the-shelf wireless devices, whereas MAClets require the software-defined radio platform USRP B200 for execution.

Software-defined Wireless Networking Software-defined networking (SDN) [43, 44, 88, 106] supports programmable network behavior in a centrally controlled manner to facilitate flexible network management. Several languages and systems can be used to program the data plane of SDN switches, such as P4 [34] or OpenFlow [161], and wireless networks [28, 64]. Programmability of wireless networks is promising especially at the PHY and MAC layers due to the dynamicity of wireless communication and the scarcity of the wireless spectrum [86, 134]. Schulz-Zander et al. have proposed OpenSDWN, an approach based on SDWN and Network Function Virtualization (NFV) [216–218]. In OpenSDWN, a virtual AP is provided for each client, where PHY and MAC layer transmission settings can be changed for each flow. Hätönen et al. [108] use intelligent edge techniques to enable SDWN on off-the-shelf APs, where virtual machines are used on AP hardware to create multiple virtual APs.

To the best of our knowledge, no existing SDWN approach facilitates the programmability of Wi-Fi functionality on off-the-shelf mobile devices, as it is supported by ReactiFi. We argue that SDWN approaches are not well-suited for the goal of programming Wi-Fi firmwares of mobile devices, because programming wireless networks is about reacting to events, and maintaining state about a device's environment (e.g., its neighbors and their distances), not only about packet processing.

Event-based Embedded Programming TinyOS [141] is a scheduler and a collection of drivers for low-power wireless embedded systems. It allows event-driven programming with nesC [98], a C language derivative. RIOT OS [18] is a microkernel-based operating system, designed to match the requirements of Internet of Things (IoT) devices. It allows thread execution with a preemptive, priority-based scheduler, but does not include integrated means to handle dataflow.

Reactive Programming for Embedded Systems Emfrp [211, 256], CFRP [240], and Hae [254] are reactive programming languages for generic embedded devices, often modeling sensor-based devices that monitor some external state. This leads to a design that focuses only on stateful reactives, i.e., the value of a sensor can always be accessed. Thus, these approaches lack built-in operations such as filtering and alternatives of events. They have been shown to be easily parallelizable [204].

Juniper [109] is an ML-like language for the Arduino platform. While it supports both stateful and stateless events, Juniper does not distinguish the two, blurring the semantics regarding

when an event fires and where the runtime has to store state. Juniper supports a dynamic dataflow graph by compiling a runtime into the target C++ code, resulting in a more complex program with a larger memory footprint. It also allows inline C++ code, similar to how ReactiFi is based on C, but both C++ and Juniper have redundancies, and the interaction between the Juniper code and the C++ code require understanding of the encodings by the Juniper compiler.

Flask [156] uses the Haskell type system to type an embedded DSL similar to ReactiFi, showing the applicability of the approach to other domains. However, since Flask targets sensor networks, the semantics of Flask are optimized for a system that allows less control, leading to a language with fewer guarantees.

CÉU [209] and Esterel [29, 84] do not directly focus on embedded devices, but bring synchronous reactive programming to soft real-time systems. Compared to ReactiFi they use a more imperative style of syntax, but their underlying semantics, i.e., processing all events in the same logical time step, is similar to ReactiFi. These languages are used in industry deployments, demonstrating the advantages over alternatives such as writing C directly.

Reactive Programming for Programmable Networks Frenetic [89], Nettle [249] and Procera [250] allow programmers to describe network policies using functional reactive programming abstractions. Compared to ReactiFi, these languages target packet forwarding on programmable network switches, by compiling to OpenFlow [161] rules. The dataflow in these languages is very specific to the semantics of OpenFlow and does not directly translate to Wi-Fi programming. Flowlog [179] adopts a database-like programming model, where internal state, represented as tables, is updated in response to incoming events. The SQL-like syntax of Flowlog hides the dataflow of applications and makes it hard to compose independent event flows without creating intermediate tables.

6.2.7 Summary

This section presented ReactiFi, a domain-specific language to facilitate programmability of embedded devices. Programmers can use a high-level reactive programming language to perceive environmental information, comprehend the device's situation, and make decisions based on the projection. Advantages of ReactiFi with respect to scheduling and memory usage, and code length by comparing two implementations of a case study in C and ReactiFi, were discussed. The empirical evaluation demonstrated the benefits of programming embedded devices with a reactive language in terms of significant improvements of throughput and latency, and with support for situation-aware embedded edge computing.

6.2.8 Appendix: Adaptive File Sharing Implementation in C

```
#define FROM_AP 0
#define TO_AP 1
#define FROM_TDLS 2

#define FC_OFFSET 6
#define DURATION_OFFSET 8
#define ADDR1_OFFSET 10
#define ADDR2_OFFSET 16
#define ADDR3_OFFSET 22
#define SEQ_CTL_OFFSET 28

#define IOV_SET 1

typedef struct {
    uint8_t addr[6];
} mac_addr_t;

typedef struct {
    uint8_t version;
    uint8_t type;
    uint8_t sub_type;
    uint8_t to_ds;
    uint8_t from_ds;
    uint8_t more_frags;
    uint8_t retry;
    uint8_t pwr_mngmt;
    uint8_t more_data;
    uint8_t protected;
    uint8_t order;
} frame_control_t;

typedef struct {
    frame_control_t *fc;
    uint16_t duration;
    mac_addr_t *src;
    mac_addr_t *dst;
    mac_addr_t *bssid;
    uint16_t seq_ctl;
    int32_t signal;
    int32_t noise;
    uint8_t ds_type;
    struct wl_info *wl;
} monitor_frame_t;

struct averages {
    int32_t avg1;
    int32_t avg2;
    mac_addr_t *addr;
    struct wl_info *wl;
};

uint8_t is_tdls = 0;
map_t addr_snr_map = 0;
int *signal_count = 0;
uint8_t MY_MAC[6] = {0};

```

```
57 // If we receive a sk_buff, we have to parse it.
58 // This is what this function is for.
59 int make_frame(monitor_frame_t *mntr, struct wl_info *wl, struct wl_rxsts
       *sts, struct sk_buff *p) {
60
61     char *raw_frame = (char *)p->data;
62
63     mntr->fc->version   = (uint8_t) (raw_frame[FC_OFFSET] & 0x03);
64     mntr->fc->type      = (uint8_t) (raw_frame[FC_OFFSET] & 0x0C) >> 2;
65     mntr->fc->sub_type  = (uint8_t) (raw_frame[FC_OFFSET] & 0xF0) >> 4;
66
67     if (mntr->fc->type != 2) {
68         return -1;
69     }
70
71     if (mntr->fc->sub_type != 0) {
72         return -1;
73     }
74
75     mntr->fc->to_ds      = (uint8_t) (raw_frame[FC_OFFSET + 1] & 0x01);
76     mntr->fc->from_ds    = (uint8_t) (raw_frame[FC_OFFSET + 1] & 0x02) >>
       1;
77     mntr->fc->more_frags = (uint8_t) (raw_frame[FC_OFFSET + 1] & 0x02) >>
       2;
78     mntr->fc->retry      = (uint8_t) (raw_frame[FC_OFFSET + 1] & 0x02) >>
       3;
79     mntr->fc->pwr_mngmt  = (uint8_t) (raw_frame[FC_OFFSET + 1] & 0x02) >>
       4;
80     mntr->fc->more_data  = (uint8_t) (raw_frame[FC_OFFSET + 1] & 0x02) >>
       5;
81     mntr->fc->protected  = (uint8_t) (raw_frame[FC_OFFSET + 1] & 0x02) >>
       6;
82     mntr->fc->order      = (uint8_t) (raw_frame[FC_OFFSET + 1] & 0x02) >>
       7;
83
84     memcpy(&mntr->duration, &raw_frame[DURATION_OFFSET], 2);
85
86     if (mntr->fc->to_ds == 0 && mntr->fc->from_ds == 1) {
87         mntr->ds_type = FROM_AP;
88         memcpy(mntr->dst, &raw_frame[ADDR1_OFFSET], 6);
89         memcpy(mntr->dst, &raw_frame[ADDR3_OFFSET], 6);
90     }
91     // REPLY TO AP
92     else if (mntr->fc->to_ds == 1 && mntr->fc->from_ds == 0) {
93         mntr->ds_type = TO_AP;
94         memcpy(mntr->dst, &raw_frame[ADDR3_OFFSET], 6);
95         memcpy(mntr->dst, &raw_frame[ADDR2_OFFSET], 6);
96     }
97
98     // REPLY TDLS
99     else if (mntr->fc->to_ds == 0 && mntr->fc->from_ds == 0) {
100         mntr->ds_type = FROM_TDLS;
101         memcpy(mntr->dst, &raw_frame[ADDR1_OFFSET], 6);
102         memcpy(mntr->dst, &raw_frame[ADDR2_OFFSET], 6);
103     }
104
105     else {
106         return -1;
```

```
107     }
108
109     memcpy(&mntr->seq_ctl, &raw_frame[SEQ_CTL_OFFSET], 2);
110
111     mntr->signal = sts->signal;
112     mntr->noise = sts->noise;
113
114     mntr->wl = wl;
115
116     return 0;
117 }
118
119 // Function for generating the key for the hashmap
120 void gen_key(char *key, uint8_t type, monitor_frame_t *input) {
121     sprintf(
122         key,                // Store in key
123         "%d %x%x%x%x%x%x",  // Format: type, space, 6 byte mac address
124         type,
125         input->src->addr[0],
126         input->src->addr[1],
127         input->src->addr[2],
128         input->src->addr[3],
129         input->src->addr[4],
130         input->src->addr[5]
131     );
132 }
133
134
135 // The first filter for filtering frames sent to this device.
136 monitor_frame_t *filter_my_frames(monitor_frame_t *input, uint8_t MY_MAC
        [6]) {
137     // Compare the 6 bytes containing the dst address with my_mac_address
138     // If it is the same, yield the next reactive.
139     // Otherwise, do nothing.
140     if (memcmp(input->dst, MY_MAC, 6) == 0) {
141         return input;
142     } else {
143         return 0;
144     }
145 }
146
147 // Aggregates SIGNAL_COUNT signals for the current address to an average
148 map_t aggregate_average(map_t state, monitor_frame_t *input, int *
        signal_count) {
149     char *key = malloc(15, 0);
150     gen_key(key, input->ds_type, input);
151
152     // Get the signals for the current address.
153     int *avg = 0;
154     int hashmap_state = hashmap_get(state, key, (void **)&avg);
155
156     // If the address is not present, we have not seen this frame yet.
157     // Initialize.
158     if (hashmap_state != MAP_OK) {
159         avg = malloc(sizeof(int), 0);
160         *avg = 0;
161     }
162
```

```
163     // Calculate the average and put the new average to the map.
164     *avg = *avg + (input->signal - *avg) / *signal_count;
165     *signal_count = *signal_count + 1;
166 
167     // Put the newly created avg to the hashmap.
168     hashmap_put(state, key, avg);
169 
170     return state;
171 }
172 
173 // Comparing signals for deciding if TDLS should be setup/destroyed.
174 struct averages *compare_signals(map_t state, monitor_frame_t *input) {
175     char *key = malloc(15, 0);
176     gen_key(key, input->ds_type, input);
177 
178     int *avg = 0;
179     int snr_avgs_state = hashmap_get(state, key, (void **)&avg);
180 
181     // If the key is not in the hashmap, something is wrong. Abort.
182     if (snr_avgs_state != MAP_OK) {
183         return 0;
184     }
185 
186     char *other_key = malloc(15, 0);
187     int *other_avg = 0;
188     switch (input->ds_type) {
189         case FROM_AP:
190         case TO_AP:
191             // If this frame if from the AP or to the AP, we compare it
    to the SNR average received directly from the other node.
192             // If it is smaller, enable TDLS.
193             gen_key(other_key, FROM_TDLS, input);
194 
195             snr_avgs_state = hashmap_get(state, other_key, (void **)&
    other_avg);
196             if (snr_avgs_state != MAP_OK) {
197                 return 0;
198             }
199 
200             break;
201 
202         case FROM_TDLS:
203             // If this frame is directly from the other node, we compare
    it to the SNR average received from the AP.
204             // If it is bigger, enable TDLS.
205             gen_key(other_key, FROM_AP, input);
206 
207             snr_avgs_state = hashmap_get(state, other_key, (void **)&
    other_avg);
208             if (snr_avgs_state != MAP_OK) {
209                 return 0;
210             }
211 
212             break;
213 
214         default:
215             return 0;
216     }
```

```
217
218     free(key);
219     free(other_key);
220
221     // Construct the return struct
222     struct averages *ret = (struct averages*) malloc(14 + sizeof(struct
        wl_info*), 0);
223     ret->avg1 = *avg;
224     ret->avg2 = *other_avg;
225     ret->addr = input->src;
226     ret->wl = input->wl;
227
228     return ret;
229 }
230
231 // Filter for enabling TDLS if it should be.
232 void enable_tdls(struct averages *avgs) {
233     if (avgs->avg1 < avgs->avg2) {
234         // Before enabling TDLS, check if it already is enabled.
235         if (is_tdls == 0) {
236             struct tdls_iovar info;
237             memset(&info, 0, sizeof(struct tdls_iovar));
238             memcpy(info.ea, avgs->addr, 6);
239
240             info.mode = TDLS_MANUAL_EP_DISCOVERY;
241             wlc_iovar_op(avgs->wl->wlc, "tdls_endpoint", 0, 0, &info,
        sizeof(struct tdls_iovar), IOV_SET, 0);
242
243             info.mode = TDLS_MANUAL_EP_CREATE;
244             wlc_iovar_op(avgs->wl->wlc, "tdls_endpoint", 0, 0, &info,
        sizeof(struct tdls_iovar), IOV_SET, 0);
245
246             is_tdls = 1;
247         }
248     }
249 }
250
251 // Filter for disabling TDLS if it should be.
252 void disable_tdls(struct averages *avgs) {
253     if (avgs->avg1 > avgs->avg2) {
254         // Before disabling TDLS, check if is enabled.
255         if (is_tdls == 1) {
256             struct tdls_iovar info;
257             memset(&info, 0, sizeof(struct tdls_iovar));
258             memcpy(info.ea, avgs->addr, 6);
259
260             info.mode = TDLS_MANUAL_EP_DELETE;
261             wlc_iovar_op(avgs->wl->wlc, "tdls_endpoint", 0, 0, &info,
        sizeof(struct tdls_iovar), IOV_SET, 0);
262
263             is_tdls = 1;
264         }
265     }
266 }
267
268 // The monitor source function.
269 // Whenever a new frame appears, we do the handling here.
```

```
270 void wl_monitor_hook(struct wl_info *wl, struct wl_rxsts *sts, struct
    sk_buff *p) {
271     if (p == 0 || p->data == 0) {
272         return;
273     }
274
275     monitor_frame_t *frm = (monitor_frame_t *) malloc(42 + sizeof(struct
    wl_info*), 0);
276     frm->fc = (frame_control_t*) malloc(11, 0);
277     frm->dst = (mac_addr_t*) malloc(6, 0);
278     frm->src = (mac_addr_t*) malloc(6, 0);
279     frm->bssid = (mac_addr_t*) malloc(6, 0);
280
281     if (make_frame(frm, wl, sts, p) != 0) {
282       goto cleanup;
283     }
284
285     monitor_frame_t *my_frames = filter_my_frames(frm, MY_MAC);
286     if (!my_frames) {
287       goto cleanup;
288     }
289
290     addr_snr_map = aggregate_average(addr_snr_map, my_frames,
    signal_count);
291
292     struct averages *tmp3 = compare_signals(addr_snr_map, frm);
293     if (!tmp3) {
294       goto cleanup;
295     }
296
297     enable_tdls(tmp3);
298     disable_tdls(tmp3);
299
300 cleanup:
301     free(frm->fc);
302     free(frm->dst);
303     free(frm->src);
304     free(frm->bssid);
305     free(frm);
306     wl_monitor(wl, sts, p);
307 }
308
309 // This is the firmware's main function.
310 // Initialize the hashmap and the counter.
311 // Enable monitor mode as the frame source.
312 void autostart(int a1) {
313     addr_snr_map = hashmap_new();
314     signal_count = malloc(sizeof(int), 0);
315     *signal_count = 1;
316
317     __attribute__((at(0x18DA30, "", CHIP_VER_BCM4339,
    FW_VER_6_37_32_RC23_34_40_r581243)))
318     __attribute__((at(0x18DB20, "", CHIP_VER_BCM4339,
    FW_VER_6_37_32_RC23_34_43_r639704)))
319     BLPatch(wl_monitor_hook, wl_monitor_hook);
320 }
```

Listing 6.5: C code of the adaptive file sharing case study

6.3 Multimodal Complex Event Processing on Mobile Devices

Embedded devices offer exciting possibilities to create new embedded edge computing applications that make use of the embedded devices' environmental information, such as user activities, location information, environmental conditions, operating system events, and network traffic, to preprocess data before transmitting it to infrastructure- or mobile devices or cloud backends. Complex event processing (CEP) is an approach for realizing situation-aware applications on embedded devices, since CEP can be used to filter, aggregate, and correlate data in a high-level language that allows developers without domain knowledge to formulate complex queries on event streams. To leverage the full potential of embedded edge computing for executing application-specific queries on heterogeneous event streams in an energy-efficient manner, *multimodal CEP* is presented, a novel approach to access and process event streams on-device. To broaden the applicability of multimodal CEP, it is able to execute queries not only on embedded devices but also on multiple modes of mobile devices, where modes in turn can be embedded devices on their own. More specific, in user space (user mode), in the operating system (kernel mode), on the Wi-Fi chip (Wi-Fi mode), and/or on a sensor hub (hub mode), with the two latter modes describing embedded devices. Multimodal CEP automatically breaks up CEP queries and selects the most suitable execution mode for the involved CEP operators, allows irrelevant components of the mobile device to be turned off or set to sleep mode, and avoids unnecessary CPU cycles, such as context switches and memory copy operations. Filter, aggregation, and correlation operators can be expressed in a high-level language that abstracts from implementation and execution details associated with the corresponding modes without requiring system-level domain-specific knowledge. Multimodal CEP enables developers to efficiently perceive information like user activities, environmental conditions, or operating system (OS) and network events. Novel situation-aware embedded edge computing applications can use this information, e.g., to trigger notifications, initiate network transmissions, or control video and audio data processing.

This section is organized as follows. Subsection 6.3.1 presents the design of the approach, including the query language. Subsection 6.3.2 describes implementation. In Subsection 6.3.3 experimental results are presented. Finally, Subsection 6.3.4 discusses related work.

Parts of this section have been published previously [104].

6.3.1 Multimodal CEP

In this section, the operator algebra, the modes of multimodal CEP, the query language, and the mode selection algorithm are presented.

CEP Operator Algebra

We use a common operator algebra for event streams to define CEP operators [132]. Formally, an *event* e is a pair (p, t) consisting of a payload p and an event timestamp t; p is from some domain $\mathcal{D}$, and t is from a discrete and totally ordered time domain $\mathcal{T}$. The validity of p is the instant t. An *event stream* E is a potentially unbounded sequence of events, ordered by t. Repeated readings of a sensor naturally compose an event stream; p consists of the measured

Table 6.2: Operator algebra for event streams [132]

Name		Description
Filter	σ_{φ}	Filters a window based on predicate (φ).
Aggregator	α	Computes aggregates (e.g., sum, average) of events in the underlying window.
Correlator	$\bowtie_{\varphi}$	Joins two windows based on predicate (φ).

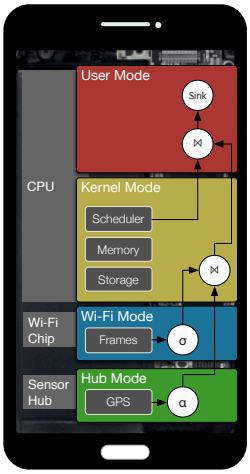


Figure 6.12: CEP operators in different modes

values (e.g., the X,Y and Z values of an accelerometer), and t is the point in time when the values are retrieved from the sensor.

Since event streams are potentially unbounded, stateful CEP operators rely on *windows* to capture the most recent event history. Windows are based either on time or on counts (e.g., the last 10 seconds or events) and move forward in a sliding (1 time unit or event) or jumping ($1 < m \leq size$ time units or events) fashion. In addition to the window operator, the algebra consists of three operators, summarized in Table 6.2. Filters, aggregators and correlators are equivalent to their counterparts in the relational algebra, but for the two stateful operators (i.e., aggregator and correlator), results are computed using the most recent set of events (i.e., the defined windows), rather than the entire event stream. The reason is that result computation over potentially infinite streams is neither meaningful nor computationally feasible (e.g., computing the sum of an infinite number of events would never produce a result).

Modes

In principle, any programmable component of a mobile device can be used to provide an execution mode for multimodal CEP. We focus on components that are useful to gather information about a mobile device's situation [25]: user activities (user processes), system events (operating system), network events (Wi-Fi chip), and physical sensor values (sensor hub). Figure 6.12 illustrates how CEP operators, as part of a CEP query, are executed in different modes: (i) user and kernel mode, both executed on the main CPU with access to

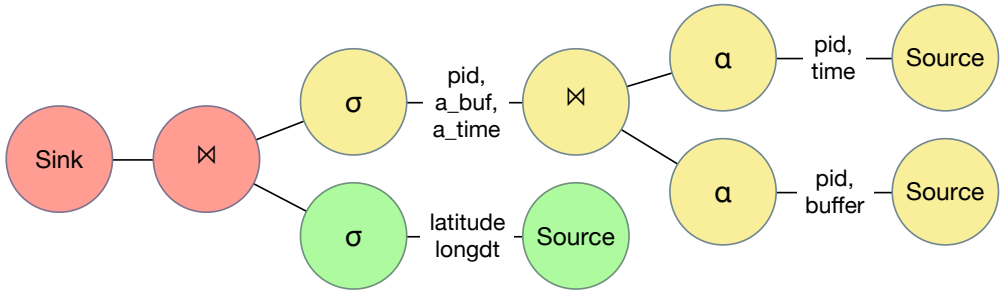


Figure 6.13: Quality-of-Experience (QoE) query example

main memory, (ii) Wi-Fi mode, executed on an ARM microcontroller of a FullMac Wi-Fi chip that usually controls Wi-Fi PHY and MAC layer protocols, and (iii) hub mode, executed on a low-power coprocessor that performs basic computations on events arriving from sensors, e.g., accelerometers or GPS trackers.

A mode is associated with a particular hardware component that can execute mode-specific binaries. A CEP operator can be executed in a particular mode if it was previously compiled for and transferred to the particular hardware component. In our approach, CEP operators are compiled on-device during runtime. CEP queries for the multimodal CEP engine are transformed into an operator tree, then the operators are assigned to suitable modes, compiled for and transferred to the corresponding hardware component.

In kernel mode, events are captured with kprobes[3]. It can be inserted in virtually any instruction in the kernel, allowing a user to break into any kernel routine and collect information non-disruptively. Berkeley Packet Filters (BPF)[4], a bytecode-based virtual machine in the Linux kernel, is used to perform, e.g., filters. In Wi-Fi mode, we use the Nexmon firmware patching framework[5] to process Wi-Fi frames on an embedded ARM processor of a FullMAC Wi-Fi chip. In hub mode, we collect and process values of a gyroscope, an accelerometer, a compass, a pressure and a proximity sensor, and a GPS sensor with an 8-bit microcontroller.

Query Language

To provide genericity and expressiveness in formulating application-specific queries on event streams without requiring domain-specific knowledge, we use a CQL-like query language based on the work of Arasu et al. [16].

```
1 SELECT AVG(len) AS len FROM
2   (SELECT * FROM kernel.sys_write WHERE pid==0)
3   WINDOW(COUNT 300 JUMP 300)
```

Listing 6.6: CQL query example

Listing 6.6 shows an an example of a query. CEP queries are represented via an operator tree: Events flow from the event sources (leaf nodes) through the operators (inner nodes) to the result sink (root). The query's input stream declaration refers to a kprobe-based event

[3]https://www.kernel.org/doc/Documentation/kprobes.txt
[4]https://www.kernel.org/doc/Documentation/networking/filter.txt
[5]https://nexmon.org

stream on the generic `write()` system call. It is mapped to a filter on the `pid` attribute, an aggregator calculating the average write buffer size, and a window aggregating the most recent 300 events.

Listing 6.7 shows a query representing a composited sensor to indicate low response times of mobile apps caused by poor network performance (i.e., high latency, low throughput) within the radius of a certain point of interest (POI), for example a railway station.

```
SELECT PID, A_TIME, A_BUF FROM
  (SELECT * FROM
    (SELECT PID, AVG(TIME) AS A_TIME
     FROM kernel.finish_task_switch
     WINDOW(COUNT 300 JUMP 300)
     GROUP BY PID)
    WINDOW(PARTITION BY PID COUNT 1) TS,
    (SELECT PID, AVG(BUFFER) AS A_BUF
     FROM kernel.tcp_cleanup_rbuf
     WINDOW(COUNT 300 JUMP 300)
     GROUP BY PID)
    WINDOW(PARTITION BY PID COUNT 1) CU
    WHERE TS.PID = CU.PID)
  WINDOW(TIME 1 S),
  hub.gps@(1 Hz)
  WINDOW(TIME 1 S)
WHERE A_TIME > LONG_DURATION AND
      A_BUF < LOW_BUFFER AND
      latitude BETWEEN l AND r AND
      longitude BETWEEN l AND u;
```

Listing 6.7: Quality-of-Experience (QoE) query example

The kernel function `tcp_cleanup_rbuf()` as a first input stream from kernel mode cleans up the "receive" buffer for full frames, before an acknowledgment is sent. It contains the number of bytes the `tcp_recvmsg()` function responsible for copying data from the kernel network stack into the user buffer has given to the user so far. Furthermore, we utilize the function `finish_task_switch()` of the Linux scheduler as a second input stream from kernel mode. It is used as a cleanup function after a task switch.

Figure 6.13 shows the corresponding operator tree with operators running in hub mode, kernel mode, and user mode. From the event sources, the event attributes `pid, time, buffer` are aggregated by two different aggregators in kernel mode. The filters are applied as close as possible to the event source, such that the conditions of the where-clause are applied as soon as these values exist. In our case, filters in kernel mode and in hub mode are used before the correlator with the predicate `pid` in user mode.

Mode Selection

We distinguish between user mode and lower-level modes, such as kernel, Wi-Fi, and hub mode. The goal of mode selection is to select modes for operators such that the number of events passed from the lower-level modes to the user mode is minimized, while the resource restrictions of these modes are met. This minimizes the overhead of inter-mode communication and reduces the load of the more power-hungry main CPU. Each of the lower-level modes has

Operator		F_{in}	F_{out}
Filter	σ_{φ}	$S[0].F_{out}$	$F_{in} \cdot sel(\varphi)$
Aggregator	$\alpha_{w,agg}$	$S[0].F_{out}$	$F_{in}/size(w)$
Correlator	$\bowtie_{w_0,w_1,\varphi}$	$S[0].F_{out} + S[1].F_{out}$	$S[0].F_{out} \cdot size(w_1) \cdot sel(\varphi) + S[1].F_{out} \cdot size(w_0) \cdot sel(\varphi)$

Table 6.3: Output event rate calculation on windows and predicate selectivity (sel)

mode specific resource constraints and hosts a dedicated set of event sources. Furthermore, the hardware architecture determines how the events are passed between operators: events from hub and Wi-Fi mode can be passed to the kernel mode or user mode, but not between hub and Wi-Fi mode; events from the kernel mode can be sent to the user mode only. Our mode selection algorithm solves a placement problem: we partition the queries' operators into candidate sets (one per mode) and decide for each operator of a candidate set whether the operator is deployed or not. In the first step, we build candidate sets for the Wi-Fi and hub mode, containing all operators that do not rely on inputs from other modes, and compute their operator placement. Afterwards, we build the candidate set for the kernel mode. This set contains all operators solely relying on events from kernel sources as well as unassigned operators from the hub and Wi-Fi candidate sets, which are (at some point) correlated with events from the kernel mode. In the last step, all operators that were not placed in one of the lower-level modes are assigned to the user mode.

The selection of to-be-placed operators from a candidate set is modeled as a variant of the 0-1 knapsack problem with mode-specific constraints. Although this problem is NP-hard, we opt for the optimal solution using an 0-1 ILP solver[6]. We limit its execution time to two seconds and use the best solution found up to that point. In all of our test cases (up to 30 operators per candidate set), the optimal solution was found within one second.

We now formally describe the placement decision problem, including the mode specific constraints. Every operator is represented by a five tuple $O_i := (C_{cpu}, C_{mem}, F_{in}, F_{out}, S)$ where C_{cpu} is the CPU cost per processed event, C_{mem} is the memory required by this operator (including attached windows and the program code), F_{in}, F_{out} are its in- and output frequencies (events/s) and S is the set of upstream operators (child nodes). C_{mem} and C_{cpu} depend on the defined window sizes, which in turn depend on an operator's input frequency (time windows). To compute these values, we traverse the operators from bottom to top and apply the operator specific formulas presented in Table 6.3 to obtain F_{in} and F_{out}. Currently, predicate selectivity is provided by the user. The remaining values are then calculated based on the frequencies.

Despite mode specific resource constraints, the placement decision for a set of candidates can be stated as an integer linear program (ILP): given a set of candidate operators $\mathcal{O} := \{O_1, \ldots, O_n\}$, we define the set of decision variables $X := \{x_1, \ldots, x_n\}$ as:

$$x_i := \begin{cases} 1, \text{ if } O_i \text{ is placed in the considered mode} \\ 0 \text{ otherwise} \end{cases}$$

[6] Sat4J, `www.sat4j.org`

The objective function to maximize is

$$\sum_{O \in \mathcal{O}} X(O) \cdot (O.F_{in} - O.F_{out})$$

It represents the number of events that are *not* transferred to another mode. $X(O)$ refers to the decision variable corresponding to O. Additionally, we define the following constraint:

$$\forall O \in \mathcal{O} : \forall O' \in O.S : X(O) - X(O') \leq 0$$

This ensures that there are no gaps in the computed placement (i.e., an operator is only placed, if all of its child operators are placed, too). We add additional constraints to reflect mode specific restrictions on CPU and memory requirements, which we describe below.

Hub/Wi-Fi Mode In these modes, all operators are executed within a single executable and thus share the available resources (processing power and memory). This is reflected by adding the following constraints to our problem formulation:

$$\sum_{O \in \mathcal{O}} X(O) \cdot O.C_{mem} \quad \leq Cap_{mem} \tag{6.1}$$

$$\sum_{O \in \mathcal{O}} X(O) \cdot (O.C_{cpu} \cdot O.F_{in}) \quad \leq Cap_{cpu} \tag{6.2}$$

They guarantee that the resource requirements of the placed operators do not exceed the mode's memory (Cap_{mem}) and processing (Cap_{cpu}) capacity. Note that the CPU costs are scaled according to the operator's input frequency to reflect the number of operator invocations during a single loop of execution. For example, if the GPS sensor is queried at 1 Hz and the accelerometer at 2 Hz, the accelerometer operators are executed twice as often as the GPS operators in a single processing loop.

Kernel Mode Queries in kernel mode are executed via BPF programs. The main difference to hub and Wi-Fi mode is that there are no global restrictions on CPU and memory usage, but an instruction limit per BPF program. A BPF program is deployed for every leaf to root path and invoked once per event. This implies that for paths joined by a correlator, both corresponding BPF programs must execute exactly the same operators for the common portion of the paths. For example, consider a query that correlates two streams and filters the results afterwards. This results in two BPF programs, one processing the left side of the join and one processing the right side. The correlator and filter can only be placed in kernel mode, if both BPF programs have the capacity to process them. To include these constraints into our problem formulation, we define $\mathcal{P}_{\mathcal{O}} := \{P_1, P_2, \ldots\}$ as the set of all longest paths for the operator candidate set $\mathcal{O}$. A path is a sequence of operators $P_i :=< O_{i,1}, \ldots, O_{i,n} >$, such that all operators are connected: $\forall j \in \{1, \ldots, n-1\} : O_{i,j} \in O_{i,j+1}.S$ and $O_{i,1}$ are attached to an event source ($O_{i,1}.S = \emptyset$). We use paths to express the instruction limit per BPF program as a constraint in our problem formulation:

$$\forall i \in \{1, \ldots, |\mathcal{P}|\} : \sum_{O \in P_i} X(O) \cdot O.C_{cpu} \leq Cap_{cpu}$$

Cap_{CPU} refers to the instruction limit per BPF program and $|\mathcal{P}|$ is the number of paths in $\mathcal{P}$. The alignment of joined paths is ensured by constraint (2).

Special care must be taken for operators that are extracted from the hub- and Wi-Fi candidate sets: they should only be executed in kernel mode, if the corresponding join is also executed in kernel mode. The rationale behind this is that if the corresponding join is processed in user land (i.e., not placed in kernel mode), all the operators should be directly processed in user land mode to avoid multiple transfers and processing costs in kernel mode. To ensure this, it is sufficient to manipulate their frequencies, such that $F_{in} - F_{out} = 0$. That is, they produce costs but do not reduce the output size unless they participate in a join.

Multiple Query Placement For multiple concurrent queries, there might be multiple paths originating at the same source. Hence, we need to check that transferring the results from all those paths is more efficient than simply transferring the events generated by the corresponding source. Consider, for example, a source delivering events with a frequency of 100 Hz. If we place three filters, each having a selectivity of 50% on this source, the expected output rate is 150 Hz, which is less efficient than simply transferring the source events at 100 Hz and evaluate the filter in user mode. To account for this case, we add artificial operators to the candidate set ($\mathcal{O}$), one per event source: $O^{src} := (0, 0, F_{in} := m \cdot F_{src}, F_{out} := F_{src}, S := \emptyset)$ where src refers to the corresponding source and m is the number of paths originating at that source. The frequencies of these artificial operators reflect the reduction of the output frequency an operator placement has to beat. We incorporate this into our model with the following constraints:

$$\forall O^{src} \in \overline{\mathcal{O}} : \forall O \in D(\mathcal{O}, src) : X(O^{src}) + X(O) = 1$$

Here, $\overline{\mathcal{O}}$ is the set of artificial operators and $D(\mathcal{O}, src)$ extracts all operators of the candidate set $\mathcal{O}$ that are directly connected to source src. These constraints act as a mutex allowing to either place the artificial operator or any combination of descendants of src. Due to constraint (2), it is sufficient to only consider the source's direct descendants here.

In hub mode, another preprocessing step is required when considering multiple queries. Since the polling frequency of a source is specified per query, we may encounter conflicting frequencies for a single source. We treat the user specified frequencies as a lower bound and execute each query with the highest specified frequency. To preserve the queries' semantics, we adjust the size of count based windows accordingly.

6.3.2 Implementation

In this section, the architecture of our multimodal CEP engine, the assembly of mode-independent CEP operators to compilable mode-specific programs, their execution model, and our mode-specific implementation are presented.

Architecture

Figure 6.14 shows the architecture of our multimodal CEP engine in Android. Mobile applications in Android run on top of the software stack, consisting of the Android kernel

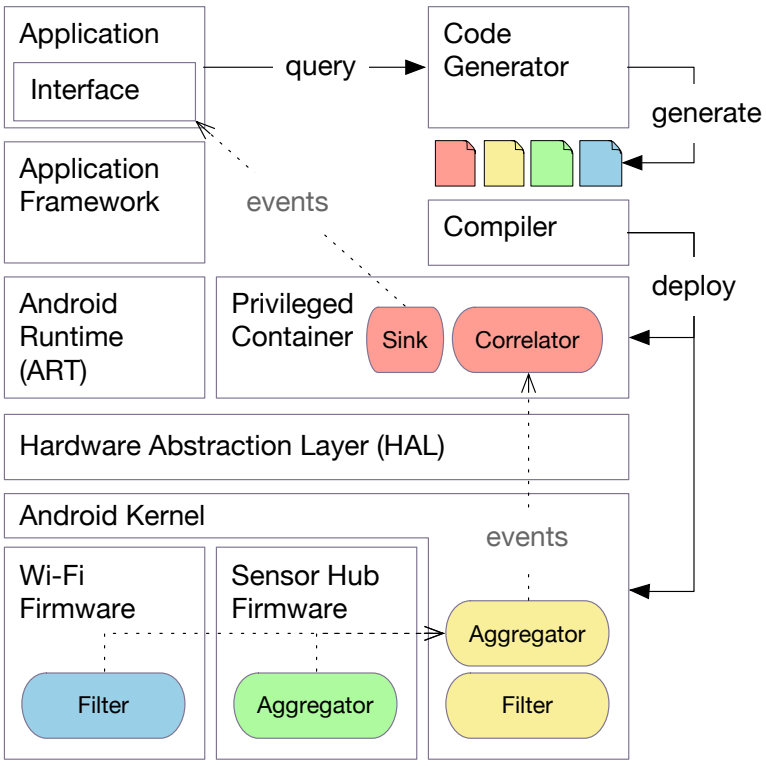
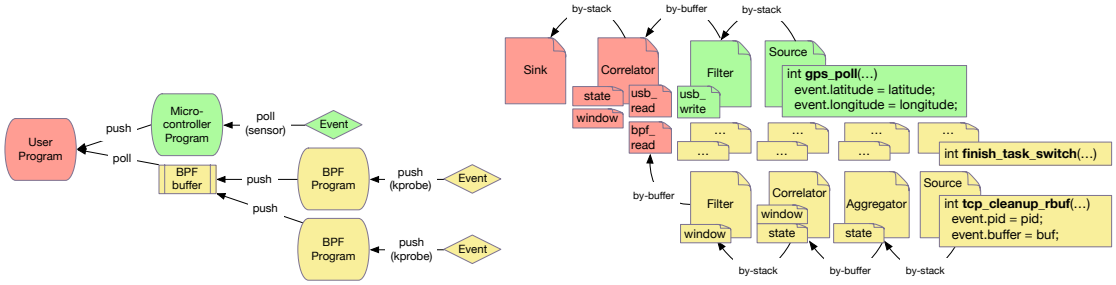


Figure 6.14: Multimodal CEP architecture in Android

with device drivers, the Hardware Abstraction Layer (HAL) with vendor-specific device implementations, the managed Android Runtime (ART) used by applications, and the Application Framework for high-level services, such as activities and notifications. Applications formulate CEP queries for the multimodal CEP engine. The multimodal CEP software stack consists of an application interface used by applications, as well as a code generator and a compiler, both running in a privileged and isolated (chrooted) container, with specific paths and libraries. The communication between applications and the container is realized via a Unix domain socket. This facilitates direct access to hardware to deploy CEP operators on the Wi-Fi chip and on the sensor hub. The multimodal CEP component generates code according to Section 6.3.2, the compiler translates the generated code into mode-specific programs and deploys the programs according to Subsection 6.3.2. Events are passed from the event sources to operators, and the event flow ends at the event sink that notifies the application. The HAL would also be a candidate to implement a mode, but since mode functions are performed similarly to the user mode on the CPU in non-privileged mode, this does not have any advantage. Thus, the HAL is bypassed. CEP operators in user mode are implemented as processes running in user mode on the CPU within the container, not as part of applications inside the ART. Our experiments showed that the throughput of our user mode is significantly higher than the throughput achieved in the ART. The throughput of the filter in user mode is about 46% higher than in the ART. The correlator is about 73%, and the aggregator is about 79% faster than in the ART, without additional power consumption. These differences are due to a larger number of in-memory data copy operations in the ART and, due to the Java garbage collection thread. Even a complex Java implementation can benefit from a generic, ring-buffer-based C implementation called via JNI.



(a) Execution model **(b)** Code generation

Figure 6.15: Execution model and code generation for the (QoE) query example

Code Generation

Each CEP operator has a defined semantics that is independent of the target architecture and its software dependencies. Therefore, CEP operators are developed as mode-independent modules, with interfaces to mode-specific operations like reading or writing buffers. These mode-independent modules are written in a subset of the C programming language. Compared to the C99 standard, this C subset is restricted as follows: (i) inlineable functions and unrollable loops: operators must be able to run in platforms like BPF, with only a predefined set of callable functions and with strong restrictions to backjumps within programs – thus, only inlined loops and functions are used for creating mode-independent CEP operator modules; (ii) no global scope and no heap memory allocations: memory sections, such as the data segment or the heap, are not available in all platforms – therefore, the internal state of an operator, e.g., the content of its input windows, cannot be stored in the global scope or in heap-allocated data; it must be read and written in mode-specific buffers, like a BPF queue.

Generic read and write functions within operators are replaced by mode-specific code during compilation. Mode-specific compilers translate operators, event sources and mode-specific dependencies to firmware patches and executable programs.

Figure 6.15b shows the code generation for event sink, event source, correlator, aggregator and filter operators as well as push and pull templates for specific transport modules. On the right side of Figure 6.15b, our code generation component weaves a producer function into the event source module. Then, it traverses the operator tree and inserts specific event producer functions as well as event queue accesses to input or output windows. Window-based operators, such as aggregators and correlators, communicate differently with their leaves, in contrast to event-based operators, such as filters and event producers. Event-based operators transport events on the program stack (by-stack), while window-based operators transport events by-buffer, i.e., they first store event data in-memory and check whether the window is in a new state, to be used as input for the next operator. Operators communicating across mode boundaries always transport events by-buffer.

To generate these source code modules, the meta-compiler must be aware of possible event sources, i.e., input streams and their attributes. This is achieved using annotated data structures in the producer functions. First, input streams are defined by producer function names, in case of Listing 6.7: `finish_task_switch()` and `tcp_cleanup_rbuf()`. Second, within these functions, an annotated function call defines the entry point for a CEP operator.

It expects an instance of an event, i.e., a pointer to a composite data type instance whose variables define the attributes of an event. This design of the input stream definitions allows developers with platform specific domain knowledge to reuse their existing implementations and to introduce CEP capabilities with minimal effort. Finally, as mentioned before, a compiler component is responsible for compiling the source code for mode-specific programs.

Execution Model

Figure 6.15a shows the execution model of multimodal CEP using the Quality-of-Experience (QoE) example in Listing 6.7. Events flow from event sources (right) to the event sink (left). Programs are concurrently executed in user mode (red), kernel mode (yellow) and hub mode (green). For the hub mode program, the microcontroller firmware is patched to periodically poll the physical sensors and push the event data to the user program. In kernel mode, two kprobe event sources push events to BPF programs. While the user as well as the microcontroller program can be run within a loop for periodic polling, BPF programs can only run per event. Furthermore, two different communication paradigms are used: push and pull. On the left, the user program is notified for new events via a blocking read from the microcontroller, while the microcontroller periodically polls the GPS sensor on the sensor hub for its sensor values. On the other hand, kernel events occur on the right and trigger a corresponding BPF program. The kernel programs push their results to a BPF queue that can be read by both user and BPF programs. Note that the mapping between operators and programs is not realized in a one-to-one manner. On the contrary, by design, one program can include as many operators as possible. Limitations for merging operators into a single program are mainly mode boundaries and resource restrictions, such as the maximum number of instructions in a BPF program or the available memory.

Modes

In this section, we present implementations of kernel mode, Wi-Fi mode, and hub mode. They are based on the generic C implementation discussed in the previous section, which is expanded to include mode-specific execution environments.

Kernel Mode In our implementation, kprobes serve as event sources. The kprobe subsystem has recently been adapted to mobile architectures, such as ARM64 processors[7], based on arbitrary software breakpoints. CEP operators run within the in-kernel BPF virtual machine as BPF programs that process incoming kprobe events. BPF allows a user land program to connect to network sockets to filter packets according to certain conditions. BPF has been added as a new feature to Linux 3.15 and has seen multiple enhancements since then, e.g., a JIT compilation feature. The combination of kprobes and BPF allows us to receive any kind of event from the kernel without modifying the source code of a particular kernel subsystem (with potentially tens of millions of lines of unfamiliar source code), thus avoiding unpredictable side effects.

[7] `https://www.linaro.org/blog/kprobes-event-tracing-armv8`

We use the BPF Compiler Collection (BCC)[8] to compile C code to BPF bytecode with the following restrictions: there is no standard library, all function calls are inlined, no loops or other return jumps are allowed, and there is a maximum number of 4,096 BPF bytecode instructions and a stack space limit of 512 bytes. Apart from BPF-specific hash tables/arrays, the stack is the only storage space available, i.e., there is no global space and no heap.

The BCC consists of an LLVM clang frontend[9] and an LLVM BPF bytecode backend. The BCC compiler uses the clang preprocessor to extract the initialization of and the accesses to BPF data structures that can be used for both the communication between BPF probes and between kernel and user land. Then, the BPF backend translates the modified C code to BPF bytecode that can be dynamically injected via the `bpf()` system call. Thus, CEP queries for the kernel mode consist of two parts: a BPF bytecode program and a user land program loading the bytecode and calling the `bpf()` system call. The user land program loads the kernel mode programs into the kernel and attaches sockets and kprobes. The BPF program can write event data into BPF-specific buffers that are accessible from the user land program via a file descriptor.

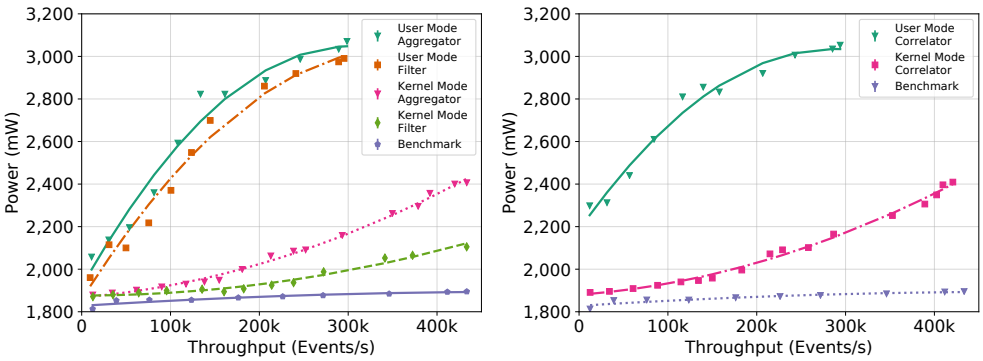
Wi-Fi Mode Wi-Fi chips of today's smartphones cannot be programmed. Therefore, our implementation of the Wi-Fi mode is based on the Nexmon firmware patching framework[10] to make firmware modifications of lower-layer frame processing on embedded ARM processors of FullMAC chips. We used the Nexus 5 smartphone that has a Broadcom BCM4339 FullMAC Wi-Fi/Bluetooth chipset and an ARM Cortex-R4 processor for controlling the dedicated MAC and PHY layer hardware. The ARM Cortex-R4 is a 32 bit RISC processor with a clock speed of 1.4 GHz.

To be able to flash and load compiled queries during runtime without disturbing Wi-Fi connectivity, we use *Position Independent Code* (PIC). Thus, queries can be compiled into separate binary files that can be loaded into arbitrary memory addresses from where we can trigger their execution. After a query has been loaded into the Wi-Fi chip, its execution is triggered by branching into the newly loaded `main` function of a PIC module. To dynamically load PIC modules containing parts of the multimodal query, we use *ioctls*.

Theoretically, the BCM4339 chip includes 768 kB SRAM and 640 kB ROM. But since we still need the standard Wi-Fi and Bluetooth functionality of this chip, and we only add new functionality, we cannot use the entire space. For program code, our patches can use about 20 kB, whereas in the standard firmware implementation without our multimodal code, about 100 kB SRAM memory is free to use. The main data structures used on the Wi-Fi chip are `sk_buffs` that are typically about 2 kB in size. Thus, queries can store about 50 `sk_buffs`. This number can be increased by performing memory optimizations, e.g., by only storing values of the `sk_buff` required by the sub-query on the Wi-Fi chip.

Hub Mode In today's smartphones, coprocessors and accelerators are used to improve performance and to decrease overall energy consumption. As a representative example, we focus on a sensor hub as a low-power coprocessor. Although most smartphone manufacturers

[8] `https://github.com/iovisor/bcc`
[9] `https://clang.llvm.org`
[10] https://nexmon.org



(a) Filter and Aggregation Operators **(b)** Correlation Operator

Figure 6.16: Power consumption in user and kernel mode processing system events

do not specify which hardware is used for such coprocessors, it is known, for example Samsung's Galaxy S4 uses a 32-bit Atmel AVR UC3 microcontroller[11] with 128 kB of flash memory, an operating frequency of 50 MHz, and 32 kB RAM[12]. Typically, current sensor hubs process events from, e.g., accelerometer, gyroscope, and compass sensors.

Since sensor hubs are usually closed-source, we have developed our own external sensor hub based on an 8-bit Atmel ATmega2560 microcontroller with a clock speed of 16 MHz, 256 kB of flash memory and 8 kB of RAM. We also have developed a second external sensor hub based on an ATmega328 microcontroller with 16 MHz clock speed, but slightly less power consumption due to reduced flash size (32 kB) and RAM (2 kB). Although both microcontrollers operate at a low clock speed comparable to other sensor hub implementations, we selected them due to their power-efficient design (26 mW/MHz and 11 mW/MHz, respectively). We use the following sensors: (i) the L3GD20H gyroscope, (ii) the LSM303 accelerometer and compass, (iii) the BMP180 MEMS pressure sensor, (iv) the VCNL4010 proximity sensor, and (v) the MTK3339 GPS sensor. The sensors (i)-(iii) are mounted on a single 10-degrees-of-freedom breakout board, whereas all other sensors reside on their own breakout board.

Our hub mode implementation contains modules for reading from the physical sensors (sensor modules), i.e., event producer implementations and event window implementations. In contrast to kernel mode, the sensor values can be stored on the heap of the processor, but due to the quite limited RAM sizes of our sensor hub (8 kB/2 kB RAM) and relatively large physical sensor values (e.g., 60 bytes for GPS), we avoid copies of physical sensor values between windows. Instead, we have implemented a dedicated reference count memory management for sensor values. If an event is needed (i.e., at least one operator uses this event), the reference count is increased. After this event is no longer needed by an operator, the reference count will be decreased. If the reference count for an event is zero, the memory will be freed and the reference will be deleted. This allows us to have about 100 filters, correlators, or aggregators with window sizes of up to 200 elements. Furthermore, a program for our sensor hub implements two basic functions: a `setup()` function used for sensor initialization, and a continuously looping main function, allowing the program to respond to physical sensor events. Since the flash memory of ATmega chips is only writable during boot and thus not at runtime of the program, the microcontroller must be flashed each time a new operator

[11] `http://www.techinsights.com/teardown.com/samsung-galaxy-s4/`
[12] `http://www.atmel.com/devices/ATUC128L4U.aspx?tab=parameters`

tree is deployed in hub mode. The communication between sensor hub and kernel is realized by a serial connection via the USB port of the smartphone. This enables us to read the corresponding device mounted in the Android kernel.

6.3.3 Experimental Evaluation

In this section, we compare kernel and user mode, a network-event benchmark comparing Wi-Fi and kernel mode, and a sensor hub evaluation.

Kernel Mode

To evaluate our kernel mode implementation, all measurements were performed on the Dragonboard 410c SoC that uses a Quad-core ARM Cortex A53 at up to 1.2 GHz per core and 1 GB RAM, running Linux 4.9 for the AArch64 architecture. The power measurements were taken at the DC input jack of the Dragonboard 410c connected to an INA219 breakout board, i.e., a power measurement SoC.

The first measurement is used to obtain a benchmark for the throughput in events/s. An event is a one-byte write to an in-memory character device, i.e., a single system call and an in-memory write. Compared to benchmarks using other kernel subsystems (e.g., sending packets via the loopback device), the CPU spents less time in the operating system. Then, we implemented a program to process these events at a specific rate (in events/s). We defined an input stream via a kprobe at `drivers/char/mem.c:write_null`, successively executed CEP operators in user and kernel mode, and measured the power consumption of the device under test. The kernel mode operators were executed by enabling BPF JIT compilation.

Due to the hard instruction limit of 4,096 instructions, we were able to compile an aggregator on a window of 300 elements and a correlator on a window of 10 elements on each input stream in kernel mode. More precisely, the correlator had an instruction size of 2,076, but its window size could not be increased due to the behavior of the BPF verifier: to ensure a safe execution of BPF programs, the verifier simulates the execution of every instruction and records the state of all registers and the program stack[13]. For a branching instruction, the verifier has to check both the true and false branch. This may result in some instructions being tested repeatedly if both branches reach the same instructions, in this case a number of 55,719 processed instructions. Overall, the verifier has a maximum of 65,536 processed instructions, which is a hard limit for our correlator implementation.

Figure 6.16 shows the results. The benchmark executed without any modifications is plotted as a baseline close to the bottom. Both the aggregator and the correlator were executed with a window size of 10 elements, no matter if they run in user mode or kernel mode. The aggregator predicate consists of a counter of all window elements. The correlator was implemented as a self-join, with the identity operation as a predicate. The kernel mode introduces an additional power consumption of 3.5% for 10k events/s, and a maximum additional power consumption of 10% for filters, and 21% for aggregators and correlators, for about 430k events/s. On the other hand, in user mode, the operators consume significantly more power, 7.5% more

[13] `https://github.com/torvalds/linux/commit/17a5267`

for filters, 12% more for aggregators, and 21% more for correlators for 10k events/s, and a maximum additional power consumption of about 35% for 300k events/s.

The difference between user mode and kernel mode is mainly due to the additional overhead of system calls, more specifically the access to the BPF-specific data structures from user land. This is also reflected by the difference between user mode aggregator and correlator. The correlator operates on two input streams, which results in twice as many accesses to the BPF data structures. Throughput and power consumption are dominated by this overhead. Other aggregator predicates do not change these results significantly. Correlator predicates with a lower selectivity have negative effects on throughput and power consumption, but do not change the overall result that it is beneficial to perform the operators in kernel mode.

Furthermore, the maximum throughput for correlators and aggregators in user mode (about 300k events/s) is approximately 30% smaller than in kernel mode (about 430k events/s). Filters can process about 600k events/s, consuming about 2,400 mW power. The benchmark results also show that the maximum throughput for both user mode and kernel mode is only a fraction of the maximum throughput of the unmodified benchmark (2.1 million events/s). This is a result of the kprobe mechanism, combined with additional CPU cycles spent to load and store event data in BPF data structures to execute CEP operators. Even a very small overhead has a strong effect at these high event rates, resulting in a throughput degradation compared to our benchmark for kernel mode operators (user mode operators) in the range of 10-15% (15-30%). To summarize, kernel correlator and aggregator consume up to 32% less power (for 140k events/s) and achieve up to 30% higher throughput compared to equivalent operators in user mode. The kernel mode filter consumes up to 32% less power (for 370k events/s) and achieves up to 52% higher throughput compared to a filter in user mode.

To summarize, the measured power consumption is proportional to the number of processed events and depends on the computational complexity of the CEP operators. On a mobile device, process- and wireless network interface-related events occur frequently. For example, a 802.11n 1x1 radio Wi-Fi interface with 150 Mbps receives 10k-100k packets/s. Applying a multimodal CEP operator, e.g., for packet inspection on a 802.11n 1x1 radio Wi-Fi interface at 10k packets/s saves 7.5% power for filters, 12% for aggregators, and 21% for correlators compared to a user land implementation. Assuming a constant rate at 10k events/s, this will lead to a 7.5%, 12%, 21% longer battery lifetime.

Wi-Fi Mode

We performed several experiments based on Nexus 5 smartphones (Broadcom BCM4339 FullMAC Wi-Fi chipset, ARM Cortex-R4 processor for controlling MAC and PHY layer, and Qualcomm Snapdragon 800 main CPU). To run experiments without interferences from the battery, we removed the battery and the charge controller from the battery, soldered wires to the charge controller, and put the controller back into the Nexus 5. Then, we performed measurements using a Monsoon High Voltage Power Monitor with a sample rate of 5 kHz and a resolution of 286 μA. The voltage was set to 4.2 Volts, which corresponds to about 92% battery capacity.

In our experiments, we used three different operators: a simple packet filter, an aggregator averaging the signal strength for a number of packets, a correlation of the packet payload

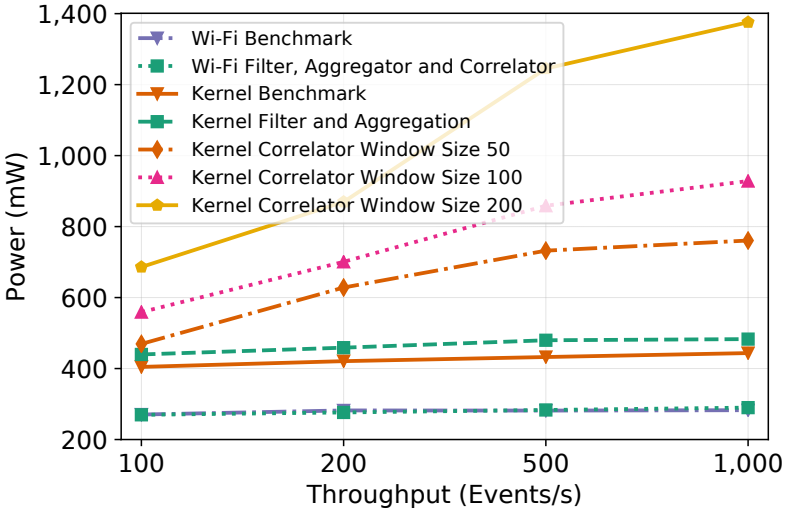
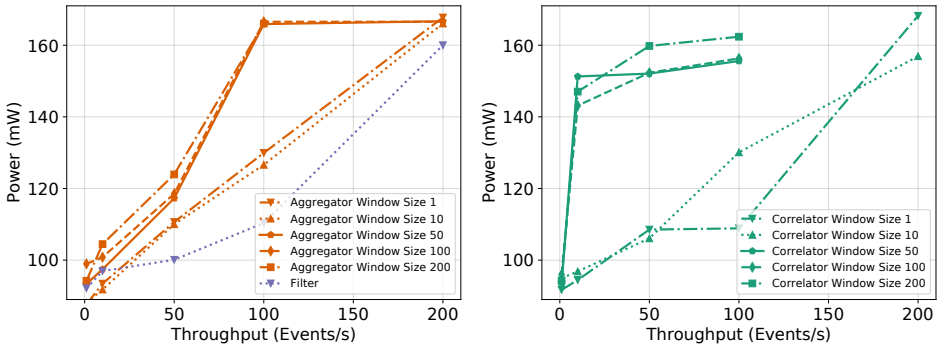


Figure 6.17: Power consumption of CEP operators in Wi-Fi and kernel mode



(a) Filter and Aggregation Operators **(b)** Correlation Operator

Figure 6.18: Power consumption of CEP operators on the ATmega328 microcontroller

size and the size of the entire Wi-Fi frame. In our aggregation and correlation tests, we used different window sizes of 1, 10, 50, 100 and 200 elements. All tests were performed with 100, 200, 500 and 1,000 ICMP echo requests per second. We compared results with the same operators performed in kernel mode.

Figure 6.17 shows the experimental results. The power consumption of operators in Wi-Fi mode was always between 265 mW and 309 mW. The power consumption of operators in kernel mode varies from 404 mW (benchmark) and 1375 mW (correlation with 200 elements). In general, the kernel mode requires between 34% more power for the benchmark and 60% more power for correlations with 50 elements than in Wi-Fi mode. In kernel mode, correlations require more power than other tests, due to the relatively high computational overhead. The high complexity of correlations is the reason why the Wi-Fi chip could not successfully perform two correlation tests with window sizes of 100 and 200 elements.

These experiments show that the potential power savings can be up to 60% if code is executed on the Wi-Fi chip instead of in the kernel. Since the Nexus 5 has a battery with a capacity of 2300 mAh, the battery would last about 12 to 19 hours longer, depending on the test. Additionally, the consumed power is always around 260 mW and 309 mW, whereas the kernel

mode shows a much higher variance due to kernel specific behaviors like context switches, which consume additional power. On the other hand, these benefits come with the problem that the Wi-Fi chip is limited in its computational capabilities, as indicated by our correlation tests with larger window sizes.

Hub Mode

The experimental setup for the evaluation of the hub mode includes the two microcontrollers used for the sensor hub, ATmega2560 and ATmega328, both operating at the same clock speed (16 MHz). The bigger ATmega2560 operates at an efficiency of 26 mW/MHz clock speed and 51.625 mW/kB main memory, while the smaller ATmega328 operates at 11 mW/MHz or 89.5 mW/kB. In other words, while the smaller ATmega328 microcontroller is more power-efficient, the ratio between memory capacity and power consumption is better for the larger ATmega2560. The microcontroller's power consumption was measured with an INA219 current sensor connected to an Arduino UNO using the I^2C bus, polled at a frequency of 2 Hz. The sensors have different sampling rates. While the GPS sensor provides new values at a rate of 1 Hz, the gyroscope provides a maximum rate of 760 Hz. Therefore, for CEP operators on input streams with low sample rates, the microcontroller is put into sleep mode when no new sensor values are available. This reduces the power consumption of the processor (ATmega328: from 170 mW to 85 mW; ATmega2560: from 408.5 mW to 194.5 mW).

Figure 6.18 shows the power consumption of the filter, aggregator and correlator executed on the ATmega328 microcontroller. The lower bound of the measured power consumption is defined by the aggregator with a window size of 1, starting at 87 mW. For all user mode operators, power consumption is about 476.7 mW (with a standard deviation of 1.9 mW). In contrast, power consumption for operators in hub mode is between 87 mW and 170 mW, which is between 19% and 36% of the main CPU's power consumption. Thus, the total power consumption savings are between 64% and 81%.

The minimum and maximum power consumption of aggregator and correlator are similar, but the maximum is reached earlier by the correlator. In general, higher window sizes correspond to longer execution times. Windows with 10 elements take about 10 milliseconds, 50 elements about 15 milliseconds, 100 elements about 20 milliseconds, and 200 elements about 30 milliseconds. Thus, a correlator with a 10-element window can run for up to 100 events/s, 50-element and 100-element windows for up to 50 events/s, and a correlator with 200-element windows with only 10 events/s.

As already mentioned, the ATmega2560 ratio between memory capacity and power consumption is better. In other words, the ATmega328 is more power-efficient at the cost of not being able to process complex operator trees. In fact, the main memory of the ATmega328 is only 25% of the size of the ATmega2560, while the flash memory is 87.5% smaller. Therefore, the window sizes for aggregators and even correlators on the ATmega2560 can be up to ten times larger than on the ATmega328.

If the microcontrollers are not put to sleep but execute the queries as fast as possible, the window sizes do not affect power consumption. Furthermore, cascading filters has no significant impact on power consumption.

6.3.4 Related Work

Multimodal CEP is related to CEP engines for mobile and resource-constrained environments, such as distributed embedded systems, wireless sensor networks, and mobile devices. For example, the μCEP engine [9] features dynamic rule updates in IoT devises. For embedded wireless devices, operating systems like TinyOS [141] and a correspondnig query processor TinyDB [155] have emerged. TinyDB, for example, focuses on reducing power by optimizing data sampling, i.e., where, when, and how often data is physically acquired and delivered to query processing operators. Additionally, several techniques for reducing the battery consumption of contextual sensing applications on mobile devices have been proposed. They range from static [168] to dynamic/adaptive [131] duty-cycling of sensors up to sophisticated processing pipelines for various sensors [152]. Existing federated CEP systems [35, 194] combine a set of heterogeneous CEP engines and provide a unified API/query language to abstract from the underlying systems. In crowdsensing scenarios, the number and reliability of available sensors vary heavily among the targeted locations. While urban areas are typically covered by more sensors than required for the targeted accuracy, the coverage in rural areas is sparse. Marjanovic et al. [160] tackle this challenge by controlling the sensors' duty-cycles to save energy by putting sensors into sleep mode while preserving sufficient sensor coverage at the same time. Their solution is based on a mobile crowdsensing publish/subscribe middleware [15] that can also be used to orchestrate the sensing process in an energy-efficient and context-aware manner. In contrast, multimodal CEP assumes that the number events processed by CEP engines (middlewares) on the main CPU (within the network or on remote servers) can be significantly reduced by detecting complex events close to the hardware and software event sources. Therefore, multimodal CEP focuses on filtering, aggregating and correlating events within the existing mobile hardware/software architecture (mode).

Multimodal CEP is related to other approaches to increase energy efficiency of mobile devices. Lentz et al. [140] optimize short-lived events in Android systems, where suspend and on transitions dominate energy consumption. The authors construct a dependency graph between woke-up threads and other components. They do not wake up all threads / components (as Android does), but a minimal set of dependent threads / components. All processes still rely on the main CPU to be executed, which can be avoided by multimodal CEP. A multimodal CEP engine can be used to complement this approach in a broad spectrum of use cases, ranging from participatory to opportunistic sensing applications, to achieve battery savings by executing queries in a more battery-friendly mode.

Moreover, our implementations of the kernel and hub mode are related to other research on sensor hubs and in-kernel processing. Several approaches [145, 222] address energy-efficiency by executing application-specific functions on a sensor hub, by automatically rewriting existing Android applications, or by providing an API for sensor-specific functions. In contrast, we propose a general approach for heterogeneous modes, including but not limited to a sensor hub. Berkeley Packet Filters are typically used for in-network filtering and processing [55], virtual network functions [6], performance monitoring, dynamic tracing and security monitoring [107]. For example, Cheng et al. [55] present a packet capturing mechanism for Android based on libpcap and BPF. The goal is to create a network packet collection framework inside the Android OS that could later be used to perform generic analyses. Compared to our use of BPF, Cheng et al. modify Android by deploying libpcap as a low-level library that can be accessed using the Java Native Interface. All these approaches use BPF or high-level language

compilers to BPF for their specific domain. In contrast, multimodal CEP processes operators in different modes and provides a generic interface for applications from different domains.

6.3.5 Summary

In this section, multimodal CEP was presented, a novel approach to process streams of events on embedded devices. In multimodal CEP, queries are formulated in a high-level language, which are then broken up and the most adequate execution mode for the involved CEP operators is selected. A multimodal CEP engine for Android devices, including three novel execution environments for CEP operators in the operating system, based on Berkeley Packet Filters, on the Wi-Fi chip, based on the Nexmon firmware patching framework, and for a custom sensor hub, leading to significant power savings of up to 81% (32%) for executing CEP operators in hub mode (kernel mode) compared to user mode, since more components of the SoC can be set to sleep mode and unnecessary CPU instructions are avoided was presented. Up to 60% power can be saved if operators are executed on the Wi-Fi chip instead of in the kernel. Throughput improvements of up to 52% in kernel mode compared to user mode were measured.

6.4 Summary

This chapter presented contributions to the field of situation-aware embedded edge computing. It was shown how the programming and data processing paradigms reactive programming and complex event processing enable novel situation-aware embedded edge computing applications that support developers to implement novel situation-aware applications.

In particular, the following contributions were presented:

- *ReactiFi:* A high-level, domain specific reactive language supporting perception, comprehension, and projection to implement actions that make novel situation-aware embedded edge computing applications possible.
- *Multimodal CEP:* Using the concept of complex event processing to support efficient filtering, aggregation and correlation for situation-aware embedded edge computing applications.

7

Case Studies for Situation-aware Edge Computing

This chapter discusses two case studies for situation-aware edge computing. Section 7.1 motivates both case studies. In Section 7.2, it is shown how situation-aware edge computing can be used to support emergency response applications. In Section 7.3, we present how situation-awareness can be used to implement transitions in the area of device edge computing. Section 7.4 summarizes this chapter.

Parts of this chapter have been published previously [23, 104, 233–235].

7.1 Motivation

Throughout this thesis, the novel concept of situation-aware edge computing was discussed. To show the feasibility, applicability, and importance of this approach, two cases studies will be presented. Since emergencies like natural disasters will continue to increase in the future due to the continuously advancing climate catastrophe, it is an imperative for modern information and communication technology to develop applications that help to cope with natural disasters, but also other types of emergencies like terrorist attacks, and to support emergency response activities with novel applications. Therefore, the first case study, presented in Section 7.2, shows how situation-aware edge computing can be used to support emergency response with three individual applications in the areas infrastructure edge computing, device edge computing, and embedded edge computing. The second case study in Section 7.3 shows how situation-awareness can be used to implement transitions to enable remote procedure calls in the area of device edge computing with intermittent and error-prone connections between mobile devices.

7.2 Situation-aware Edge Computing in Emergency Response Applications

Unfortunately, societies around the world are hit by various emergencies every year, be it terrorist attacks, infrastructure collapses such as failing bridges or crumbling power grids, or of course, natural disasters. Due to the advancing climate catastrophe, such events, but above all natural disasters, will continue to increase in the future. Thus, it is a moral imperative that modern information and communications technology (ICT) helps to cope

with such emergencies. Fortunately, many scientists and engineers are working on various aspects of emergency management using ICT. One area that is particularly affected in many emergencies is communication. Particularly in natural disasters, the communication infrastructure often fails for several days. However, there are promising approaches in this area to cope with such events. Disruption-Tolerant Networking (DTN) makes it possible to establish a rudimentary communication or data exchange without a communication infrastructure[21, 191]. Furthermore, alternative radio technologies such as LoRa are discussed, which can be used in case of an emergency to establish a temporary communication infrastructure spanning a large area [111]. However, since during emergencies communication infrastructure is often not available, centralized cloud services cannot be reached either, i.e., many functionalities, applications, and services cannot be supported. As a result, rescue helpers have to use other information sources like social media, e.g., to gather information about how many people are within an affected area [80]. Finally, many devices (e.g., smartphones, notebooks, IoT devices, embedded devices, and infrastructure devices) have computing and storage resources, but these cannot be straightforwardly used with current computing paradigms.

What all of these approaches lack, however, is situation-awareness. By implementing situation-aware applications and leveraging the strengths and benefits of edge computing, ICT can support emergency management, help affected people and rescue helpers make informed decisions, and complete critical tasks utilizing the computational resources. In the following sections, three applications are introduced. In Section 7.2.1, 5G infrastructure devices are enhanced to facilitate infrastructure edge computing during emergencies so that rescue helpers can utilize the computational capabilities to achieve situation-awareness for informed decision-making, e.g., during the planning of a firefighting operation. The infrastructure devices themselves are also using a situation-aware approach to detect emergencies. In Section 7.2.2, an approach is presented that detects faces in and extracts them from images on mobile devices while, depending on the situation, specific devices for particular computation steps are chosen to spread the load fairly across all mobile devices so that sparse resource consumption is achieved. Using this information, rescue helpers can in turn identify missing people and coordinate search operations. Finally, Section 7.2.3 utilizes embedded devices to provide rescue helpers with information regarding the number of affected people in proximity and their locations in an energy-preserving manner, so rescue helpers can comprehend the situation with respect to how many people are missing and make informed decision as to where to intensify the search for missing people.

The remainder of this section is organized as follows. Subsections 7.2.1, 7.2.2 and 7.2.3 introduce and evaluate situation-aware approaches to cope with emergencies in the areas of infrastructure edge computing, device edge computing, and embedded edge computing. Subsection 7.2.4 concludes this section.

Parts of this section have been published previously [23, 104, 233, 235].

7.2.1 Situation-aware Infrastructure Edge Computing

With the advent of 5G, network access will be available ubiquitously. High bandwidth using wireless communications is usually achieved with higher frequencies. Higher frequencies, however, result in lower penetration (e.g., of buildings) and a lower transmission range. Thus, to achieve high bandwidths in 5G networks, so-called 5G small cells have to be deployed

throughout a city [247]. This requirement leads to new opportunities for telecommunications companies to equip the facilities (e.g., small cells) they roll out for 5G Internet access with computing capabilities, such that a dense network of infrastructure devices becomes available. In this section, a novel approach is presented that uses smart street lights equipped with Internet access, but also infrastructure devices for infrastructure edge computing. The source code of this proof-of-concept implementation is available under a permissive license[1]. While smart street lights have many benefits, such as weather and air quality measurement plus public 5G Internet access and tourist information during non-emergency times, they can functionally transition into something different in the event of an emergency. Here, additional sensors are activated and sensing can be performed with a higher sampling rate, plus live video feeds can be used. The sensors (especially a camera) can be used to automatically gather awareness of critical situations prior to an emergency alarm. Furthermore, a hidden covert channel is used for secure communication with a command center during emergencies. All of this is done directly on an infrastructure device without having to send all the sensor data to the cloud. This enables us to provide situation-awareness directly at the infrastructure edge, allowing rescue helpers to make and implement decisions for further actions based on the current situation.

This subsection is organized as follows. In Paragraph 7.2.1, presents the design. Paragraph 7.2.1 discusses how situation-awareness can be achieved and how information with respect to the current situation can help rescue helpers coping with emergencies.

Smart Street Lights as Infrastructure Devices

Smart street lights are an emerging technology for digital cities. Manufacturers are currently exploring their possibilities and chances. The main goal of smart street lights is to provide additional features to citizens like Internet access or supply additional information from local sensors. Thus, a smart street light should contain the following components beyond a light source: temperature-, humidity-, CO_2-, particulates- and light sensors and motion detectors should be available as a basic set of sensors. In endangered areas with high crime rates, cameras should also be available in the lamp posts. It is also suggested including batteries and solar panels to be able to power the smart street light during emergencies. In case of an emergency, a visual guidance system should be available as well, to help to direct crowds remotely. Further, smart street lights should contain a dedicated mesh network that connects all smart street lights to a city-wide network. Finally, a smart street light should contain a device that allows executing computations at the infrastructure edge.

As a proof-of-concept implementation, a prototype of a smart street light was built. A prototypical smart street light from Schréder[2], one of the largest street light vendors globally was used. Their product *Shuffle* is a modular street light, with some capabilities required already available. First, their modular street light provides the street light itself. As of writing this work, there are no 5G small cell access points freely availble for purchase. Thus, the Wi-Fi access point (AP) provided by the vendor of the street light was used to simulate Internet access though the smart street light. For special purpose solutions or other use cases, radio links such as LoRa, ZigBee, or Sigfox can easily be added to the setup. Furthermore,

[1] `https://github.com/stg-tud/emergencity_demo`
[2] `https://www.schreder.com/`

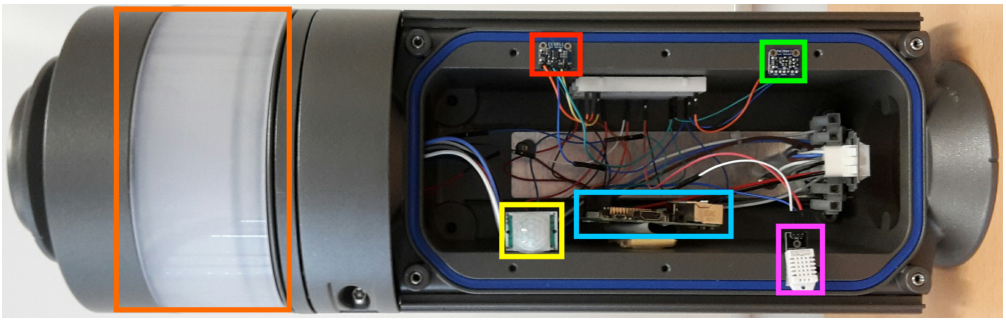


Figure 7.1: Sensors of the smart street light; orange: visual guidance system; yellow: motion sensor; red: CCS811 CO_2 sensor; blue: Raspberry Pi; green: TSL2561 light sensor; pink: AM2302 temperature and humidity sensor

Schréder also provides a camera module that was used for building the proposed prototype. Additionally, a visual guidance system is available (orange in Figure 7.1).

A set of sensors, as shown in Figure 7.1 was integrated: an Aosong AM2302 temperature and humidity sensor[3] (pink), a TAOS TSL2561 light sensor[4] (green) that can sense infra-red light and the remaining broadband spectrum. To sense CO_2, an AMS CCS811 gas sensor[5] (red), a HC-SR501 PIR motion sensor[6] (yellow) is used. Fianlly, a Raspberry Pi (blue) is used as the infrastructure device to execute computations.

Achieving Situation-awareness using Smart Street Lights

When no emergency happens or has happened (from here on called everyday mode), the smart street light functions as a light emitter and Internet AP for the inhabitants of the city. In everyday mode, the smart street light has two tasks. First, it should provide information that can be used by citizens to achieve situation-awareness on their own. Thus, the existing sensors can be used to show the temperature, humidity, air quality, and other available sensor data. The second task of the lantern is to perceive the information provided by the sensors directly at the infrastructure edge in order to comprehend its own situation. By then projecting the future situation, it can be decided whether local warnings about different events have to be issued. For example, the CO_2 and particulates sensor could be used to inform the local residents to close their windows due to poor air quality. Temperature and humidity sensors are also valuable for local climate information and behavioral suggestions like giving drinking advice during hot periods. Motion detectors and light sensors can be used to only turn on the light if it is dark enough and motion, i.e., people that require light, are detected. Additionally, the light sensor can also be used to dim or brighten the light according to the current situation. The optional camera should, for privacy and data protection reasons, not record or stream data anywhere, but evaluate the taken pictures directly at the infrastructure edge, i.e., within the street light.

[3] https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf
[4] https://cdn-shop.adafruit.com/datasheets/TSL2561.pdf
[5] https://cdn.sparkfun.com/assets/learn_tutorials/1/4/3/CCS811_Datasheet-DS000459.pdf
[6] https://www.mpja.com/download/31227sc.pdf

With the infrastructure edge computing capabilities and above described situation-awareness of the smart street light, it can detect emergencies and transition to a second mode, which is called emergency mode. The functionality of the smart street light can then change to support emergency management with information regarding the emergency that can be perceived by rescue helpers to achieve situation-awareness. They can plan and coordinate their missions based on the situation. For example, a smart light may have detected a fire situation, but the fire department may not know where the fire is taking place. Perceiving the information coming from a particular smart street light and comprehend the situation can help to project the future fire situation with respect to the size, intensity, and location of the fire. Furthermore, the camera that only evaluates taken pictures locally during everyday mode, will now send the pictures to a central command center using a dedicated mesh-based communication channel, again, supporting the rescue helpers to handle the situation. Besides providing information for citizens and rescue helpers, the smart street lamp can also execute own actions based on the made situation-aware decisions. For example, the visual guidance system can be used to direct people. A red signal could mean that no one should go this way, and green signal, on the other hand, could indicate a safe direction.

Additionally, during emergencies, the different rescue helpers and authorities on site need a common communication channel to coordinate recovery actions. This can be provided by the mentioned city-wide wireless mesh network. However, this communication channel must not interfere with other legitimate radio spectra to prevent disturbances, and it should be secure and robust against attackers, e.g., during cyber-attacks or an act of terrorism. Therefore, a transition of the wireless mesh network to address these properties is proposed. To build the inter-light mesh network, ESP8266 boards (ESPs from now on) for providing a reliable mesh network between the street lights are emplyed. To implement the communication channel for helping authorities, some features of the ESPs boards were exploited. The crystal of the ESPs oscillates with a frequency of either 40 MHz or 26 MHz. To adjust this frequency to the frequency required for emitting IEEE 802.11 compliant electromagnetic signals, the ESPs have two Phase-Locked Loops (PLLs) (i) RFPLL for adjusting the center frequency, and (ii) BBPLL for adjusting the bandwidth frequency and other peripherals. BBPLL is adjusted using registers in the CPU, since they have to be adjusted depending on whether the ESP has a crystal with 40 MHz or 26 MHz. This means that this factor can be set at runtime, resulting in altered bandwidths. Since in the IC of a Wi-Fi chip different components are used to ensure a steady and clean radio signal like high- and low-pass filters, the factor for BBPLL cannot be set arbitrarily. After analyzing the possible values, a bandwidth of 8 MHz was chosen, which is not IEEE 802.11 compliant anymore. This results in Wi-Fi signal only decodable by other ESPs with the same BBPLL values, or software-defined radios. In fact, off-the-shelf Wi-Fi devices cannot even see the Wi-Fi frames. Another advantage of this approach is that the bandwidth of the Wi-Fi spectrum are effectively shrunk, reducing the overlapping space of neighboring channels and thus reducing interferences with them. Therefore, we also favor 8 MHz over the other possible bandwidth of 16 MHz. Since a layer of encryption on the MAC layer was used, a secure and hidden communication channel that reduces the interferences with legitimate radio spectra was created.

7.2.2 Situation-aware Device Edge Computing

One key problem during many emergencies is that communication infrastructure is disturbed or even completely destroyed. Thus, it is not often not possible to offload tasks to cloud computing facilities. In such scenarios, device edge computing can play a key role in providing compute resources within the affected area, enabling at least basic computations in cases where nothing else is usable. However, especially in emergency situations, power is a sparse resource and should be preserved as much as possible. To this end, situation-awareness can play a key role in (i) providing appropriate device edge computation facilities and (ii) at the same time reduce load on mobile devices and preserve power. One application that benefits from being situation-aware is the task of detecting faces in images. To support the search for missing people during an emergency, it is sufficient to rely on the extracted imagery of a face instead of the entire image. It can even be advantageous to send only the face, since the load on the network is significantly reduced, but the information provides the same result. A situation-aware device edge computing solution can extract faces from images that help rescuers to find and identify people and at the same time preserve resources in the network and reduce the overall load of the network but also of computing and power resources. To execute this task in a situation-aware manner, the OPPLOAD framework was used, which is discussed in Section 5.2.

Paragraph 7.2.2 briefly describes the workflow that is used to detect faces in images using mobile devices. In Paragraph 7.2.2, the experimental setup is presented, and Paragraph 7.2.2 discusses the results.

Detecting Faces in Images

For situation-aware device edge computing in emergency response application, the algorithm presented by Lampe et al. [137] for detecting faces in images on smartphones was adopted, which consists of five steps: (i) image denoising, (ii) upscaling the image by 10%, (iii) cropping the image by 10%, (iv) gray scaling the image, and finally (v) detecting faces.

Experimental Setup

This application was evaluated using 30 emulated and randomly moving nodes. Since OPPLOAD is designed to use the best available worker for executing a task in the current situation, workers were assigned with different capabilities (see Section 5.2.3): 20% of the workers were capable with no constraints, 40% were also capable but with less energy reserves, 30% could execute the task, but with limited capabilities (like little available memory) and 10% were not capable to execute the task at all. Workers announce their capabilities to the network so that clients can perceive this information. Based on the perception, clients then can comprehend the situation with respect to the capabilities of surrounding workers and project the future situation as to which of the available workers is suited best for the given task. Based on this projection the clients decide on which worker the task will be offloaded to. Furthermore, the behavior with 5 and 10 clients that offload tasks at the same time in the network at the start of an experiment were evaluated, which can lead to workers executing multiple tasks simultaneously. To simulate an IEEE 802.11g network, which is still widely used especially in

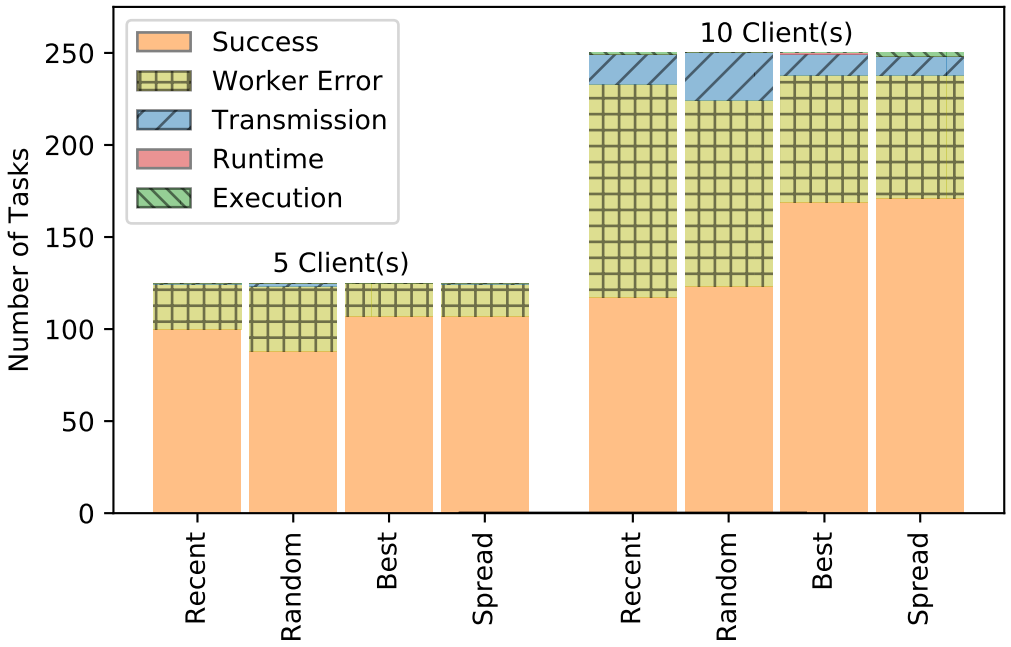


Figure 7.2: Final workflow states, by number of active clients in JiT mode.

countries of the global south, with a bandwidth of 54 Mbit/s, a basic range model for the Wi-Fi nodes with 40 meters of range was used. The mobility model was configured for 30 nodes, walking randomly in an area of about 1.7 km^2 at a speed between 0.8 m/s and 1.9 m/s or rest for up to 60 seconds, which corresponds to human walking speed. This setup leads to relatively small mesh networks that are appearing and disappearing during the execution of the experiment. Overall, 200 experiments were executed.

Evaluation

Figure 7.2 shows the final states of the workflows executed in the specific scenarios, where the bars are grouped by the number of clients per experiment and worker assignment. The y-axis shows the number of tasks in a particular state. The first case is a successful workflow (*Success*), where a workflow was offloaded, all tasks could be executed, and the result arrived at the client. Second, OPPLOAD performed as intended, but errors occurred (*Worker Error*) and the client could successfully be informed about this error. However, since an opportunistic network was emulated, errors are expected, as bundle delivery in opportunistic networks cannot be guaranteed. OPPLOAD handles three classes of error. (i) *Task execution errors* occur during the execution of the task itself, e.g., due to an exception in the code of the task. (ii) *Worker selection errors* occur if no worker for the subsequent task can be found. (iii) *worker calling errors* occur for example when the worker is no longer capable to execute the task. An experiment stopped in the *Transmission* state, if a task was transmitted to the next worker, but not received until the end of the experiment, e.g., if the recipient cannot be reached due to network fragmentation. Due to experiment abortion while OPPLOAD was in a runtime state or a worker executed the task itself, it is denoted as *Runtime* and *Execution*.

Experiments using the recent assignment mode (for the different assignment modes, see

Table 7.1: Average runtimes of tasks in mobile JiT scenarios in seconds.

Assign.	Exec. (s)	Runt. (s)	Transm. (s)	Total (s)
Recent	8.7 (0.64)	5.0 (1.89)	269.0 (336.37)	282.8 (338.91)
Random	8.9 (1.02)	5.0 (1.84)	254.9 (300.75)	268.8 (303.61)
Best	8.9 (0.62)	5.2 (1.80)	135.5 (191.26)	149.6 (193.68)
Spread	8.9 (0.68)	5.1 (1.95)	234.2 (300.75)	248.1 (303.61)

Section 5.2.3) have the lowest success rates, which is due to the fact that workers are selected that are far away and the offers arrive late. Using a random worker increases the number of successes slightly. Using the best worker available, all tests were either successful or the client was informed about an error when 5 clients are used. The spreading approach is as good as using the best worker in terms of successful workflows or errors returned in time. The fact that even using the best worker does not lead to 100% successful executions is due to the worker capabilities and the transmission time in opportunistic networks. A worker updates its capabilities after executing a task, which can lead to the situation that another task is offloaded to the worker, even though it is not capable anymore. The falsely assigned worker will decline task execution and inform the client.

Table 7.1 shows the average workflow runtimes over all experiments. It is evident that using the spread algorithm gives better results than random assignment and using a recent worker. Note that the transmission times (and thus also the total times) have a rather high standard deviation. This is due to the mobility of the nodes and potentially disappearing links between two nodes, resulting in re-transmissions. These increase the time, whereas many transmissions are successful within the first try, reducing the mean transmission time.

7.2.3 Situation-aware Embedded Edge Computing

In the area of embedded edge computing, embedded devices can contribute to emergency response applications by providing information regarding the current situation. Using the sensing capabilities of embedded devices and the pre-processing capabilities presented in Chapter 6, the gathered information is then used for further situation-aware applications for emergency response. For example, rescue helpers can coordinate search operations based on the number of people in certain areas. But the embedded devices themselves also need to be situation-aware, for example to decide whether specific sensors should be used or not, e.g., to save energy, since energy may be in short supply during an emergency. In this section, wireless devices in the vicinity of a particular embedded device are counted to estimate the number of people in a certain area. In emergency cases, knowing how many people are in an affected area can be live-saving [20, 56, 212]. First and foremost, information is provided that can be used in downstream processes to obtain situation-awareness at a higher level, for example in the creation of a situation picture. However, to do this efficiently and in a resource-saving manner, the embedded devices themselves have to be aware of their particular situation, i.e., to only use GPS positions if the device is outdoors, because the accuracy of the GPS sensor is too low indoors anyway.

Paragraph 7.2.3 presents the code that is used to count nearby devices and report the information to rescue helpers. Paragraph 7.2.3 describes the experiment setup to evaluate this approach and Paragraph 7.2.3 discusses results from the evaluation.

Counting Nearby Devices

Listing 7.1 shows the code for this application that is implemented using ReactiFi, the language presented in Section 6.2. The program counts MAC addresses of Wi-Fi management frames collected in monitor mode on all Wi-Fi channels and sends the number of addresses to the host operating system every 200 ms.

```
1  Source(Timer(10ms))
2    .fold({ 0 })((channel, time) ⇒ { (channel % 20) + 1 })
3    .observe(SwitchChannel)
4  val addrs = Source(Monitor)
5    .filter(frame ⇒ { frame.type == MANAGEMENT })
6    .map(frame ⇒ { frame.src })
7  val timer = Source(Timer(200ms))
8  val count = fold({ hashset_new() })(
9                timer → (acc, time) ⇒ { hashset_new() },
10               addrs → (acc, addr) ⇒ { hashset_add(acc, addr) })
11             .map(p ⇒ { sizeof(p) })
12 timer.snapshot(count.change(0))
13   .map(tuple ⇒ { tuple.snd })
14   .observe(SendToOS)
```

Listing 7.1: ReactiFi program for counting nearby devices

Lines 1-3 switch through all Wi-Fi channels. A time-based event source that triggers an event every 10 ms is used. The *fold* reactive aggregates state given an initial value, i.e., it counts how often the source has triggered an event to compute the channel that should be selected. This *fold* reactive then propagates the channel number to the *observe* reactive that executes the *SwitchChannel* side effect. *SwitchChannel* switches to the provided Wi-Fi channel. Line 4 shows a reactive *addrs* that is derived from a chain of reactives to gather all Wi-Fi frames in monitor mode, filter out all non-management frames (line 5), and then project the source MAC address field using *map* (line 6). A second timer to report the number of distinct addresses seen within the last 200 ms (line 7) is used. The total count is obtained through a *fold* reactive with multiple triggering conditions, one on *timer* and one on *addrs* (lines 8-10). Both parts refer to the same aggregated state, initialized with an empty set (*hashset_new()*). When *timer* triggers, the set is reset to become empty again, and when *addrs* triggers, the source addresses are collected in that set. When both trigger, the functions are executed in their defined order. The *count* (line 12) maps the accumulated set to its size. A *snapshot* reactive reads the current value of *fold* reactives, when another reactive triggers, i.e., in line 12, a snapshot of *count* is taken, when *timer* triggers. However, the *timer* reactive also resets *count* to zero. To report the count before the *timer* reactive triggers, *change* was used. Using *change* produces a reactive that reports both the current and the previous value, such that programmers may reason about what happened before the current time step. The parameter passed to *change* is used as the initial value of *fold*, e.g., zero in the example. In lines 13 and 14, the *map* reactive then propagates the device count to the *observe* reactive that executes the *SendToOS* side effect. *SendToOS* sends data back to the operating system.

```
SELECT * FROM
  (SELECT src, COUNT(*) FROM wifi.framecount)
  WINDOW(TIME 1 S JUMP 1S),
  GPS@(1 Hz) WINDOW(TIME 1 S JUMP 1S);
```

Listing 7.2: Correlating Wi-Fi management frames and GPS positions

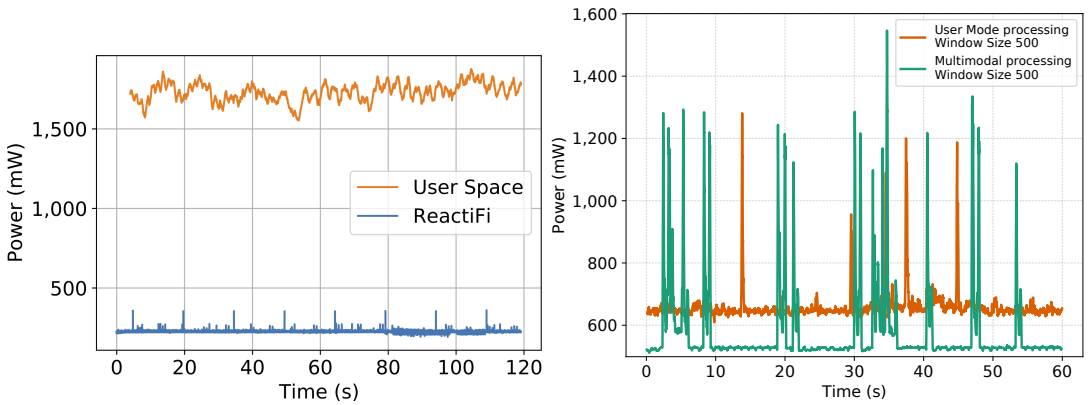
```
SELECT * FROM
  (SELECT * FROM hub.proximity@(10 Hz)
   WHERE ambient > a_threshold)
  WINDOW(TIME 1 S JUMP 100 MS),
  (SELECT * FROM hub.magnet@(10 Hz)
   WHERE SQRT(x^2 + y^2 + z^2) < m_threshold)
  WINDOW(TIME 1 S JUMP 100 MS);
```

Listing 7.3: Outdoor detection query

From here on, further processing is executed using the CEP engine presented in Section 6.3. Listing 7.2 shows our multimodal CEP query. First, it gathers the device count gathered from Listing 7.1 in a window of 1 second, i.e., the first part contains all devices seen within the last second. Second, the GPS sensor on the sensor hub is polled at a frequency of 1 Hz. The final correlation combines the current GPS position and the Wi-Fi management frame counts. Since GPS is unreliable and power-intensive indoors, the query should only be executed outdoors. At this point, the embedded device itself has to be aware of its situation with respect to whether it is indoors or outdoors. This is achieved by an outdoor detection query shown in Listing 7.3. First, it perceives all information read from the light sensor. Next consists of two filters are used to comprehend the situation and to project the future situation whether the device is outdoors or not. The first one filters all values that are below a certain threshold, indicating that the ambient light is probably too dark. The second filter sieves all values above a threshold produced by the magnetometer, indicating that the magnetic field is relatively weak, which usually holds for indoors. Based on this situation, the decision is made whether to enable the GPS sensor or not.

Experimental Setup

Experiments with two different implementations were performed: (i) where all frames and sensor values are pumped into user space, (ii) with all sensor values are pre-processed on different embedded devices within a mobile phone, i.e., the coprocessor that perceived the data. Using implementation (i), all queries were executed in user mode, while both sensor hub and Wi-Fi chip sent all data to the process in user mode. To simulate a reporting of counted devices to rescue helpers, both implementations periodically sent the result of the queries to a server using an HTTP POST method. With option (ii), the outdoor detection query and the GPS part of Listing 7.2 were executed on the sensor hub, while only the result of the correlation was sent to the process in user mode. The Wi-Fi management frame filter and the aggregation of listing 7.1 were executed on the Wi-Fi chip. The final correlator operated in user mode. A Nexus 5 smartphone in a controlled environment to identify the number of devices around the phone was used. Five other devices were distributed around the room so that a constant and verifiable count could always be guaranteed.



(a) Power consumption of the implementation from Listing 7.1 executed with and without ReactiFi

(b) Power consumption of the query from Listing 7.2 executed with and without multimodal CEP

Figure 7.3: Power consumption of the above presented code

Evaluation

In the first part, the power consumption of the ReactiFi code shown in Listing 7.1 is compared when run on the Wi-Fi chip versus being executed in user space. Figure 7.3a shows the average power consumption for both implementations over all runs. The code executed in user space requires about $1700\,\text{mW}$ power on the average with $630\,\text{mW}$ standard deviation. The ReactiFi program uses only $225\,\text{mW}$, thus, achieving $87\,\%$ improvement compared to the user space implementation. The peaks every $15\,\text{s}$ are due to the LTE interface trying to connect to a network even if no SIM card is present.

Figure 7.3b shows the power consumption of the Nexus 5 smartphone and the sensor hub. For this experiment, the ReactiFi code is always executed on the Wi-Fi chip, whereas the CEP queries are either executed entirely in user mode or on the respective coprocessors. The user mode requires a ground truth of about 660 mW. As soon as all Wi-Fi management frames are counted, the subsequent processing and communication with the sensor hub and the user mode requires more power, i.e., up to 1.3 W. If the queries are executed by multimodal CEP, the test hardware only needs about 564 mW while executing the query on the sensor hub and the Wi-Fi chip, respectively. Additionally, during the multimodal tests, the power consumption shows more peaks, due to the more complex communication between Wi-Fi chip, sensor hub, and user mode. The average power consumption during the multimodal tests is about 578 mW, while the same queries executed in user mode require 668 mW on the average, which is an increase of 13%.

7.2.4 Summary

This section presented situation-aware edge computing for emergency response applications. It was identified that by applying the concept of situation-awareness to emergency response applications based on infrastructure edge computing, device edge computing, and embedded edge computing approaches, rescue helpers and people affected by emergencies are provided with useful information and applications that help to handle emergencies.

In particular, three applications where presented:

- *Smart Street Lights as Infrastructure Devices* can be used to inform citizens about emergencies based on the situation, but also to provide information to rescue helpers that they can use to make informed decisions.
- *Detecting Faces in Images* showed an approach that utilizes the strengths of mobile devices to detect and extract faces from images. Due to the fact that it is a situation-aware approach, the load of this task is distributed fairly among all mobile devices and thus resources are preserved.
- *Counting Nearby Devices* utilizes embedded devices to count people in proximity efficiently so that rescue helpers can coordinate search operations based on the information.

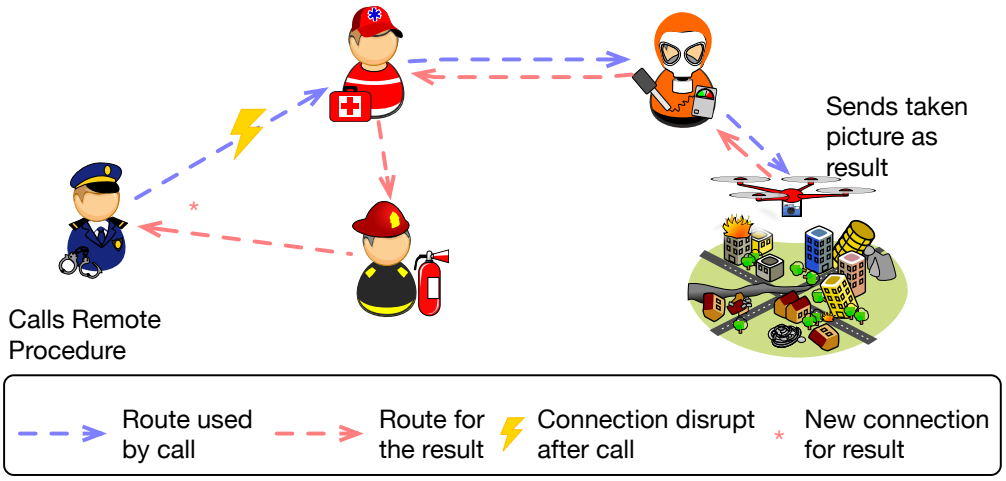


Figure 7.4: Calling a remote procedure in a DTN disaster scenario

7.3 Situation-awareness for Transitions at the Edge

To enable device edge computing, paradigms are needed that allow mobile devices to execute procedures on surrounding mobile devices. Remote Procedure Calls (RPC) offer this possibility following the client-server paradigm [32], where clients call procedures that are implemented on remote servers. However, none of the existing RPC implementations are designed to work properly in device edge computing environments, where devices are mobile, resulting in potentially periodic, intermittent, and prone to disruptions. One possible way to cope with these challenges is using the Disruption-Tolerant Networking (DTN) paradigm [77, 162].

The combined use of RPCs and DTNs can provide great services for various scenarios. For example, enabling mobile devices with limited computation resources can leverage the power of surrounding mobile devices for computations in rural regions, e.g., in India [126] and Australia [97], where no telecommunication infrastructure exists and where people cannot communicate using mobile devices. Another example is to support emergency response applications by providing civilians and rescue helpers computational resources. For example, quadcopters could offer a procedure that takes a picture with a mounted camera at a particular geographical location and returns it over the network. Then, rescue helpers could request an overview image via an RPC to a quadcopter while performing other tasks until the file arrives over a DTN connection using nodes of other rescue helpers or citizens as relay nodes. This example is illustrated in Figure 7.4, where the call takes the blue route, but the result arrives over the red route due to the connection loss illustrated by the yellow lightning symbol. This might take longer, but without DTN the call could not be made at all.

However, to support this kind of device edge computing it is necessary to ensure that the possibility of getting a result of a called remote procedure is maximized. To ensure that the most suitable transport protocol is used, the transition paradigm is proposed to be applied. Here, mobile devices can dynamically transition between various transport protocols to maximize the number of successful remote procedure calls. The question arises, however, on what basis to decide which transport protocol to use. The question arises, however, on what basis to decide which transport protocol to use. It is argued that a situation-aware

approach provides the appropriate basis for decision-making, since this implies that the most suitable protocol is used in each situation.

In this section, DTN-RPC is presented, a new approach to provide RPCs for DTN environments by utilizing the concept of transitions. DTN-RPC relies on several components that allow RPCs to be executed in DTNs. First, it implements transitions between DTN and Non-DTN transport protocols for transmitting data between clients and servers, called *transparent mode*. The decision which protocol to use is implemented in a situation-aware manner. Clients perceive information regarding their network state (e.g., whether a server can be reached directly using a lookup-mechanism) and comprehend this information to gather their situation. Further, they project their future situation with respect to whether a server is reachable using an end-to-end transport protocol. Based on the projection, it is decided if a DTN or a Non-DTN protocol should be used to call the procedure. The client also has to be aware of incomplete information as network lookups in DTNs are error-prone and might result in inaccurate results like wrongly reporting that a server is reachable. Servers perform a similar approach to achieve situation-awareness to decide which protocol should be used to return the result of the called procedure to the client. The second component used by DTN-RPC to allow RPCs in DTNs is the usage of two distinct channels, control- and data channels, to cope with potentially short contact durations in DTN, where it is potentially impossible to transmit large amounts of data using an end-to-end protocol. Using this separation, it is ensured that both meta-data (e.g., procedure names and parameters) and payload (e.g., images that are to be used for a particular procedure) are transmitted using the most suitable transport protocol. Third, explicit and implicit modes for server addressing are supported that enabled the possibility of increasing the rate of successful RPCs by allowing arbitrary servers to respond to called procedure. Finally, to only execute procedures that the called server is able to, servers perceive information from its own system sensors and predicates from clients to comprehend its situation with respect to its current capabilities and available resources. Based on this comprehension, the server will project whether a procedure will be executed or not, again increasing the success rate of called procedures, because clients will be informed of calls that are not executed so that it can call this procedure on another server.

The open-source implementation of DTN-RPC[7] is based on Serval [21, 94–96], an open-source, disruption-tolerant wireless ad-hoc networking system. Experimental results obtained within the network emulation framework CORE indicate that the measured CPU and network overheads for DTN-RPC are reasonably low, so that DTN-RPC can be executed on mobile devices, and that the round-trip times and the number of successful RPCs are highly satisfactory in dynamically changing network topologies with unreliable connectivity due to the transparent transitions between transport protocols.

The remainder of this section is organized as follows. The design of DTN-RPC, the proposed framework to provide transitions at the edge, will be presented in Subsection 7.3.1. Implementation issues will be discussed in Subsection 7.3.2 and experimental results will be shown in Subsection 7.3.3. Existing work on RPCs will be examined in Subsection 7.3.4. Subsection 7.2.4 concludes this section.

Parts of this section have been published previously [234].

[7] https://github.com/adur1990/DTN-RPC

7.3.1 DTN-RPC

Fundamental Considerations

There are several differences between RPCs in traditional networks and RPCs in DTN.

In conventional RPC implementations, errors are handled, for example, if the connection between client and server is lost. In DTN, it is not certain whether a call even reaches its destination. Thus, errors in DTN can only be handled in a few situations, since error reports could just not arrive and the client would not notice that the call was not successful. The server, on the other hand, would have to spend computational overhead while trying to inform the client about the error. Furthermore, disruptions and poor connection quality make it impossible to support real-time communication or to guarantee a predefined quality of service in DTN. Common RPCs are location transparent. For this purpose, stubs or proxy functions exist to handle communication via the network. In DTN, a call will explicitly be executed remotely, and it is expected that there will be networking overhead when executing a remote procedure. In several RPC implementations, the client has to register at the server before calling a procedure. Since in DTN the address of a server is typically not known, client registration is not possible. Traditional RPC servers either announce the procedures they offer or there exists a lookup service where clients can find information about which server offers which procedure. In DTN, server announcements might not reach or lookup services might not be available for clients when needed.

Control and Data Channels

DTN is often used in mobile mesh and ad-hoc networks where the network topology changes frequently. This can lead to short contact durations between nodes where it is impossible to transmit large amounts of data. Due to this restriction, two separate communication channels are introduced in DTN-RPC: the *control* and the *data* channel.

The *control* channel is responsible for transmitting meta-data, such as the procedure name and the parameters, from client to server, and possible results from server to client. The *control* channel supports two modes to address remote servers, *explicit* and *implicit* (*any* or *all*), as described below.

Explicit If the address of a server is known and the server is reachable, DTN-RPC will choose the *explicit* mode and try to establish an end-to-end connection to the server.

Implicit (any or all) If the address of a server is not known, but potential servers are reachable, both the *any* and *all* modes (summarized as the *implicit* mode) are used to broadcast a call. In the *any* mode, the client waits for exactly one response. This is helpful if it is known that servers exist that offer a particular procedure, but it does not matter which server responds. The first arriving response will be accepted. In the *all* mode, the client will wait for as many answers as possible until its internal timeout occurs. This is useful in scenarios where the quality of the results varies with the executing machine (e.g., GPU support, different algorithms), where different answers should be combined (e.g., to

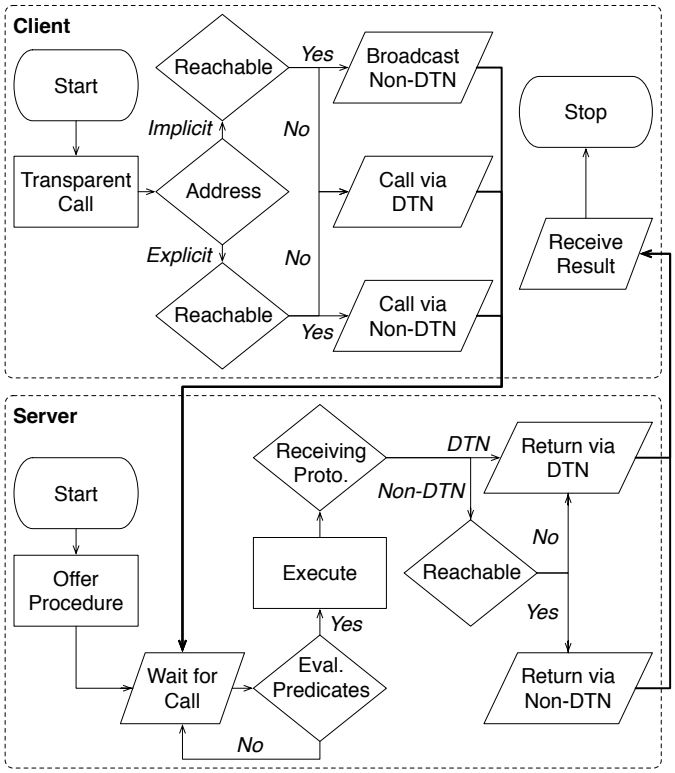


Figure 7.5: DTN-RPC flowchart for client and server

implement aggregate functions that return a value across all items in the results set), or is influenced by other factors such as geolocation (e.g., sensor readings, taking a picture).

The payload of the *control* channel packets must not exceed the payload size of the underlying transport protocol to keep the data on the network as small as possible.

The *data* channel transports larger amounts of data from client to server and vice versa. It is used if a file is required as a parameter for a particular call. The transport of the payload in the *data* channel is always performed via DTN. The transport of the meta-data in the *control* channel is explained below.

Transparency

In both *explicit* and *implicit* addressing modes, the *control* channel of DTN-RPC supports *Non-DTN* and *DTN* transport protocols and automatically switches between them for performing a procedure call, as explained below.

Non-DTN vs. DTN As illustrated in Figure 7.5, if the server is reachable in the *explicit* mode, DTN-RPC will use a Non-DTN transport protocol to call the server. If the server is not reachable, the call will be issued using a DTN protocol.

After having called a remote procedure in the *explicit* mode, the client waits for the response using the same transport protocol that was used to call the procedure. If the connection is interrupted, the client additionally waits for results that arrive via a DTN protocol.

After having successfully executed a received call, the server checks whether the explicit control channel on which the call was received via a Non-DTN protocol is still available, as shown in Figure 7.5. If the channel is not available anymore, the result will be sent via a DTN protocol. The DTN-RPC server does not attempt to reestablish a Non-DTN connection, since it is unlikely that a reconnection is successful if one of the nodes has physically moved out of the network's reach. If the call was received via a DTN protocol, the server also uses a DTN protocol for its response.

Since the *implicit* modes use broadcast addresses to call procedures, a different transport protocol has to be used than in the *explicit* mode, because reliable point-to-point transport protocols like TCP do no support broadcast packets. Since a server availability check in a broadcast scenario would imply communication between multiple nodes, which would add additional delays, a call just gets broadcasted without any prior availability checks. If a timeout occurs and no result arrives, the call is performed via a DTN protocol.

Transparent DTN-RPC is designed to automatically select the most suitable transport protocol in any given scenario. In the *transparent* transport method, both client and server are designed to make all the above discussed decisions without any user interaction.

Offering and Executing Calls

Offering a remote procedure as shown in Figure 7.5 is a two steps process. The first step is the declaration of the procedure as a prototype in a configuration file to tell the server which procedures are available for execution. Second, the implementation has to be provided as an external executable written in any programming language.

The parameters of an incoming call are passed in the order they were received to the external program that then executes the procedure. After the procedure finishes, the result is returned to the server that marshals the result and prepares the result to send it back to the client.

Typically, computational resources and battery lifetimes of nodes in DTN are limited. To avoid the execution of calls that would consume too many resources regarding a server's curent state, a server can decide whether a remote procedure should be accepted. For this purpose, we define particular predicates per server, e.g., thresholds for resource constraints (number of concurrent processes, remaining battery life etc.) or available hardware like sensors (also shown in Figure 7.5). The server checks whether defined predicates are satisfied. If at least one requirement is not met, the procedure will not be executed.

Furthermore, each call can provide its own requirements that also have to be checked by the server. For example, some calls should only be executed on non-moving nodes, or require special sensor hardware or extensive resources, such as disk space or RAM. Therefore, there is a two-stage predicate check per server: the first one is the general server acceptance check, and the second one is call-specific and evaluated after having passed the first check.

7.3.2 Implementation

The implementation of DTN-RPC is based on the Serval Project [94–96]. Serval is centered around a suite of protocols designed to allow infrastructure-independent communications. The Serval Mesh Protocols abstract from lower-layer protocols, such as IP, UDP, Wi-Fi or others. Serval's real-time packet-switched protocol is the Mesh Datagram Protocol (MDP), which can be compared to UDP/IP, but uses SIDs (Subscriber ID, the public key of an asymmetric elliptic curve key pair) instead of IP addresses, and includes encryption, authentication and integrity features by default. To route packets, MDP uses a protocol inspired by OLSR [59] and B.A.T.M.A.N. [124] for both node discovery and maintaining a routing table, which facilitates multi-hop routing of packets. On top of MDP, the Mesh Streaming Protocol (MSP) provides reliable data streaming, similar to TCP. Finally, Rhizome is a simple store-and-forward protocol defining files as *bundles*. Intended as the DTN protocol of Serval, Rhizome uses an epidemic routing protocol to transmit files hop-by-hop from source to destination. Rhizome is purposely agnostic of the transport protocols below it, requires no routing table and focuses on single-hop communications, with multi-hop communications emerging as a natural consequence of *bundles* replicating among nodes. DTN-RPC uses MDP, MSP, and Rhizome to handle different situations and addressing modes.

An in-depth experimental evaluation of Serval's DTN aspects for various network setups and usage patterns in the previous work [21] was conducted. The results have indicated that Serval is capable of handling extreme conditions such as saturated networks or many-hop transmissions in a satisfactory manner. It has also been shown that Serval works well in realistic scenarios, where the topology changes over time and users have different requirements. Thus, Serval is an elaborate and ready-to-use software for DTN and mesh networks.

For programmers, an API is offered that can be used to develop programs using the DTN-RPC library to execute procedures on remote devices in DTN environments.

Calling a Remote Procedure Transparently

To call a remote procedure transparently, a single function is required that is part of the offered DTN-RPC API. This function has five parameters: the server address, the name of the called remote procedure, the number of parameters of the procedure, the parameters themselves and the execution requirements discussed in Section 7.3.1. The mode to be used is determined by the first parameter of this API function call.

Explicit If the parameter is a valid address the call will be issued via Serval's MSP, if the server is available. A routing table is built in an ad-hoc manner. If the address of the server can be found in this routing table, this particular server is reachable. While waiting for the result, the client checks periodically whether the connection is still alive. If the connection terminates, the client starts a Rhizome DTN listener.

Implicit The modes *any* and *all* are used if the address is the *ANY* address provided by Serval for *any* or the broadcast address for *all*. Since Serval's MSP supports point-to-point communication only, it is not possible to send data to the broadcast address. Therefore, *any* and *all* use Serval's MDP.

Since a reachability test is not possible for broadcasts, the procedure will be called without any checks. Because delivery is uncertain, the client sends a call every second until at least one server responds or a timeout occurs. If an acknowledgment arrives, the threshold for the timeout is increased. Only if the new timeout occurs, the client will additionally start a Rhizome DTN listener and wait for the result via DTN.

The difference between the modes *any* and *all* is the number of results. In the first case, the client stops listening as soon as the first result arrives. In the second case, the client waits for as many results as possible, but at least for one.

Returning the Result Transparently

While executing the called procedure, the server does not check periodically whether the client is still reachable. Instead, this check is done once when the response is ready to be sent. If the call arrived via MSP or MDP, but the connection is broken, or the client is not reachable, sending will fail, and the server will send the result via Rhizome.

7.3.3 Experimental Evaluation

In this section, an experimental evaluation of DTN-RPC for different network topologies and in various configurations is presented. Due to the lack of comparable RPC implementations that can handle disruptive networks, DTN-RPC is not compared against other approaches. A comparison with widespread software solutions such as JSON-RPC or SOAP would be unfair, since they would fail each time the network connection is lost.

Test Setup

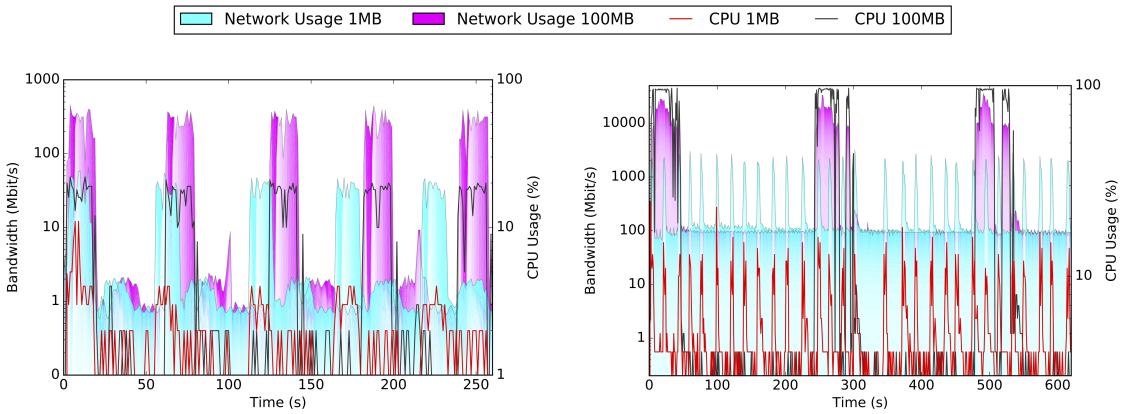
The evaluation of DTN-RPC is based on the open source network emulation framework CORE[8]. Compared to protocol simulations, CORE can run DTN-RPC without modifications in a more realistic Linux environment. All tests are performed on a 64-core AMD Opteron 6376 CPU with 256 Gigabyte RAM, emulating up to 64 virtual nodes at the same time.

Measurements Standard Unix tools are used to measure system properties with a time resolution of one second. For CPU statistics, *pidstat*[9] is used, and the Serval and DTN-RPC processes are monitored from within a node. Network usage is measured from within the nodes on every network interface for Serval and DTN-RPC using a custom Python script based

[8] `https://www.nrl.navy.mil/itd/ncs/products/core`
[9] `http://sebastien.godard.pagesperso-orange.fr`

Table 7.2: Topologies

Name	# Nodes	Description
Hub	28	All nodes connected to each other
Chained	32	Pair-wise connected
Islands	64	Partitioned islands with dynamic links in between



(a) *Chained* topology **(b)** *Hub* topology

Figure 7.6: Bandwidth and CPU usage for *1 MB* and *100 MB* in different topologies

on *libpcap*[10]. To monitor the behavior of DTN-RPC, metrics such as call times, round-trip times, and logging functions were implemented and integrated into the binary.

Network Topologies Three network topologies are considered, as shown in Table 7.2.

Hub The *Hub* topology connects 28 nodes with each other. As shown in the previous work [21], the Hub topology is challenging for Serval and thus also for DTN-RPC due to the high number of direct neighbors, all using bandwidth and flooding each other with status information. Therefore, the *Hub* topology helps to investigate whether DTN-RPC can handle RPCs when the network is under heavy load.

Chained The *Chained* topology consists of a chain of 32 nodes, 31 hops from the first to the last node. Typically, network connections over the Internet require less than 16 hops. In a DTN mesh network, more hops might be needed for messages to reach their destination.

Islands The *Islands* topology represents a partitioned, dynamic network with 64 nodes. At the beginning, there are 4 islands each containing 16 nodes. The 16 nodes per island are connected randomly with each other, creating an ad-hoc mesh network. Then, four different behaviors can occur randomly every 60 seconds: two islands are connected, two connected islands are disconnected, all islands are connected, or all islands are disconnected resulting in the original state.

[10] `http://www.tcpdump.org`

Network Connections DTN-RPC adds a new layer of abstraction to the Serval networking stack. Although Serval can cope with several degraded networking scenarios, DTN-RPC is only evaluated in situations where network connections are completely lost, because this is the most challenging situation in DTN. Network degradation and bandwidth limitations would only lead to higher delays, but not break DTN-RPC itself.

Test Sets and Modes The remote procedure used implements an echo service. It is called with three different test sets: (i) *0 MB*, where no file is used; (ii) *1 MB*, where a file of 1 megabyte is transmitted; (iii) *100 MB*, where a file of 100 megabytes is sent.

Additionally, all tests are executed in 10 different modes: *explicit*, *any* and *all* via Rhizome; *explicit*, *any* and *all* via MDP; *explicit*, *any* and *all* transparently and *explicit* via MSP.

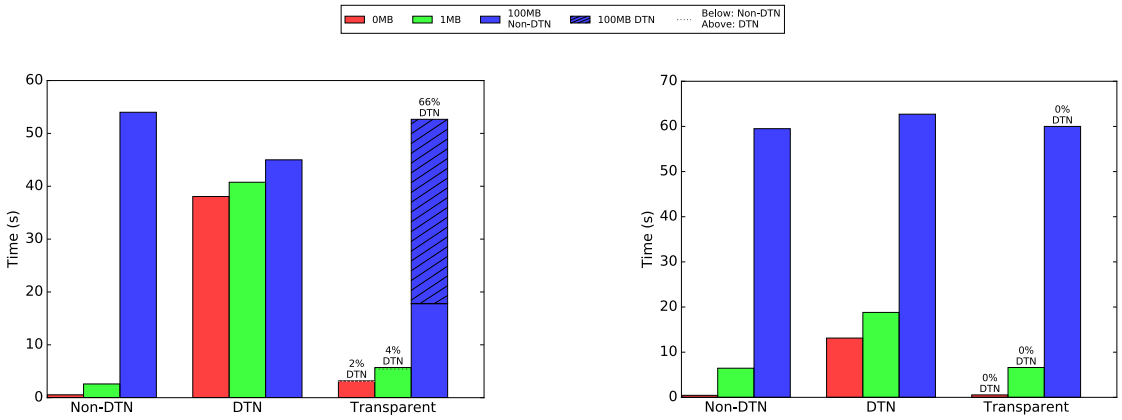
Servers Since the successful execution of remote procedures in DTN depends on the number and distribution of servers, every test in *Hub* and *Islands* is executed twice, first with 5% of the nodes as servers and second with 50%. In *Chained*, the goal is to determine how DTN-RPC performs if the call has to travel a long distance. Thus, only one server and one client at the opposite ends of the chain are needed.

In each test setup, the procedure is called 30 times to get reliable results. The acknowledgement from the server has to arrive within 30 seconds on the explicit channel. After the acknowledgement, the client waits an additional 90 seconds for the result. If within these 90 seconds no results arrived, the procedure is called via DTN, which has an additional 90 seconds to finish. After the client has received the result or all timeouts are reached, the next procedure will be called.

Since the evaluation is concerned with the overhead and the performance of DTN-RPC, the possibility of DTN-RPC to perform predicate checks to decide whether a remote procedure should be accepted has been disabled in our experiments.

Fundamental Properties In *Hub* where each node is a single hop away from all other nodes and Serval uses broadcast packets to announce meta-data, each node produces a flood of data that is sent to all neighbors. Thus, both the CPU usage and the network load in *Hub* are always higher than in the corresponding tests in *Chained* or *Islands*, due to the high number of direct neighbors. Furthermore, DTN-RPC does not only use the API, but also the networking stack and the communication mechanisms provided by Serval. Thus, DTN-RPC cannot be measured separately, but only together with other Serval traffic.

Similar to the network usage, the CPU utilization has to be measured not only for DTN-RPC, but also for Serval running on a node. The evaluation of the CPU usage shows that the CPU consumption of DTN-RPC is negligible with about 1% in heavy load situations. However, the Serval process has a higher CPU usage, since Rhizome computes a hash for each file sent. The larger the file, the more time-consuming the hash computation becomes. DTN-RPC, on the other hand, is independent of file sizes, because it simply issues a call to the Rhizome API, which leads to the described 1% CPU utilization increase in the DTN-RPC process. Therefore, since the CPU utilization is dominated by Rhizome, in the experiments below it is always based on the Serval process.



(a) *Chained* topology **(b)** *Hub* topology

Figure 7.7: Round trip times in different topologies

Network Performance

For the *0 MB* tests in the *Chained* topology, the overall network load averages at about 2 Mbit/second for each of the three transport protocols (MDP, MSP and Rhizome). This is true for all three modes, *explicit*, *any* and *all*. Since DTN-RPC uses only a single packet for calling the remote procedure and returning the result in *0 MB*, these packets get lost in the overall network load that is produced by Serval exchanging meta-data and therefore not plotted in Figure 7.6.

During the *1 MB* and *100 MB* test sets, the network load increases up to 70 Mbit/s for *1 MB* and up to 500 Mbit/s for *100 MB*, as indicated by the blue and red graph of Figure 7.6a, in which the stacked bandwidth for all network interfaces together with the CPU usage in a logarithmic scale for 5 calls with the *100 MB* test set and 30 calls with the *1 MB* test set is shown. In the *1 MB* and *100 MB* calls, a file always has to be transmitted via the Rhizome DTN for calling the remote procedure and receiving the result. The difference between the *1 MB* and *100 MB* calls is due to the different file sizes.

The *Hub* topology shows a similar behavior, as illustrated by Figure 7.6b, where the stacked bandwidth for all network interfaces together with the CPU usage in a logarithmic scale for 3 calls with the *100 MB* test set and 30 calls with the *1 MB* test set is shown. The main difference is that the *Hub* topology suffers from the problems discussed in Section 7.3.3. The overall network usage for the *1 MB* test sets exceeds 1,000 Mbit/s (blue graph) and 10,000 Mbit/s for the *100 MB* test sets (red graph).

Comparing the bandwidth consumption to the previous results [21], DTN-RPC does not add any measurable network traffic to the traffic produced by Serval, and thus can handle scenarios where the network has a high bandwidth usage well.

CPU Usage

As shown in Figures 7.6a and 7.6b, CPU usage highly correlates with network usage. Since CPU usage in the *0 MB* tests does not exceed 1% after the initial discovery phase, it is not

plotted in Figures 7.6a and 7.6b. For the *1 MB* tests, the maximum is at about 2% up to 3% (red line) and up to 20% for the *100 MB* tests (black line) in the *Chained* topology.

In the *Hub* topology, the behavior is comparable to the *Chained* topology, with the difference that the CPU usage is generally higher. In the *1 MB* tests, the CPU usage increases up to about 10% and for the *100 MB* tests up to 90% during the sending phase. This relatively high CPU consumption happens only while a hash of a file is computed and the file is inserted into the Rhizome store, and thus only during a relatively short time period. As already mentioned, the CPU usage of DTN-RPC does not exceed 1%.

Round Trip Times

To measure the round-trip times (RTTs), only the *Chained* and *Hub* topologies are considered, since the *Islands* topology would not give any credible results. RTT is only used to indicate the time that is needed to transmit the payload through the network to be sure no additional delays are introduced by DTN-RPC. The execution of a procedure typically takes longer to finish than the implemented echo service.

As shown in Figure 7.7a, the *0 MB* tests in *Chained* called by MDP or MSP (i.e., Non-DTN) are executed within a second. As the files grow, the RTT increases.

In the DTN tests, the RTTs are similar, regardless of the file size. Due to the fact that in DTN the control channel as well as the data channel are transferred via Rhizome, both server and client have to wait for two files. Therefore, all tests take about 40 seconds.

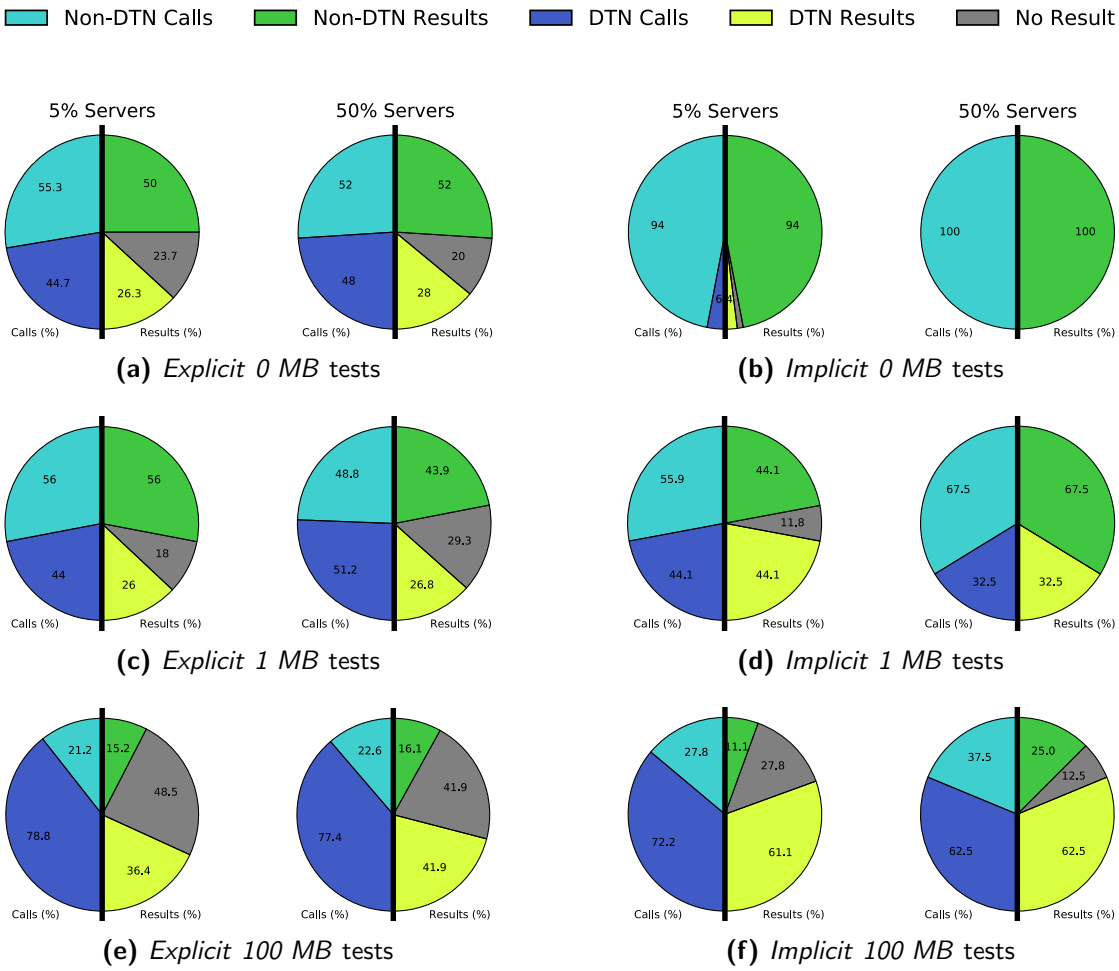
Transparent calls are slower than the calls via MDP or MSP for the *0 MB* and *1 MB* tests. Some calls are issued via MDP or MSP, while others are executed via Rhizome, as explained in Section 7.3.1. The illustrated RTTs are averaged over 30 calls, including the slower Rhizome calls. Furthermore, the time it takes to wait until the transport protocol will be switched is also part of the RTT. Therefore, the *transparent* tests are slower than the corresponding *explicit* tests, but faster than the DTN tests. Since all *100 MB* tests are issued using Rhizome and the switch time is included in the RTT, the time it takes for finishing is higher than for MDP or MSP.

As shown in Figure 7.7b, the RTTs for tests in *Hub* do not differ much from the tests in *Chained*. The only difference is that the *0 MB* and *1 MB* tests are faster in *Hub*, because all nodes are only one hop away from each other.

To summarize, DTN-RPC can execute remote procedures satisfactorily fast. The fallback method using Rhizome is slower, but still can get a result back to the client within an acceptable time, even if the files are large.

Transparency Behavior

In this section, it is examined how DTN-RPC behaves in the dynamic *Islands* topology with different numbers of available servers. The figures below show how many of a total of 30 procedures are called using Non-DTN or DTN, respectively, in terms of percentage values. The left half of the pie charts represents outgoing calls and the right half incoming results.



(a) *Explicit 0 MB* tests **(b)** *Implicit 0 MB* tests

(c) *Explicit 1 MB* tests **(d)** *Implicit 1 MB* tests

(e) *Explicit 100 MB* tests **(f)** *Implicit 100 MB* tests

Figure 7.8: Percentages of procedures called and results returned via Non-DTN and DTN for *100 MB* in the *Islands* topology

Since the *Islands* topology consists of 4 islands with 16 nodes that merge and separate over time, it is possible that not all results arrive within 210 seconds at the client if the call was issued in *explicit* mode, especially in tests with only 5% servers. Additionally, as the file size increases, the transmission time increases too, and the number of successful calls decreases as expected, as indicated by Figure 7.8a, Figure 7.8c, and Figure 7.8e. Furthermore, it is evident that some results arrive via MDP or MSP (i.e., Non-DTN), others only via Rhizome (i.e., DTN). There are two reasons. First, it is possible that a call is issued successfully using MSP, but the route from the server to the client gets lost because the islands have separated. Then, the result is sent via Rhizome and arrives after the islands have merged again. Second, the client cannot establish a connection to the server at all, because the islands are not connected. The procedure will be called using Rhizome and the client will wait via Rhizome for the result. Even if some results do not arrive in the *explicit* mode, the DTN protocol helps to improve the number of successful calls, as shown in Figure 7.8. 41.9% of the results in the *explicit* tests with the *100 MB* test set with 50% of the nodes as servers arrive via Rhizome, and in 41.9% of the tests, no result arrives. In the *implicit* tests with the *100 MB* test set with only 5% of the nodes as servers, 61.1% of the results arrive via Rhizome, and only 27.8% of the results do not arrive at all.

Figures 7.8b, 7.8d and 7.8f show *implicit* tests in the *Islands* topology for three different file sizes with different numbers of servers. It is evident that the *implicit* mode increases the number of successful calls in every situation compared to the *explicit* tests. Due to the dynamically changing *Islands* topology and the relatively short contact durations, it is still possible that not all results arrive in *100 MB*. For the *explicit* calls, the more servers are available, the more results arrive.

The number of missing results can be decreased if the contact duration is increased or the waiting time for results is increased. Furthermore, more elaborate remote procedures require a lot more time to finish than the simple echo service used in the evaluation. Therefore, the waiting time for results of up to 210 seconds in the experiments should be increased in production environments, since it might be possible that a result arrives after hours at the client via DTN.

To summarize, the *transparent* mode helps to improve the probability of receiving results in dynamically changing network topologies like *Islands*. Furthermore, the *transparent* mode can deliver results where a traditional RPC would not lead to any response due missing network connections. Finally, if the waiting time for results is adequately large, the probability of receiving results increases, because when a DTN protocol is used, results do not get lost, but simply are not transmitted via a direct connection to the receiving node. Therefore, given sufficient time, results will always reach their destinations.

7.3.4 Related Work

Tu and Stewart [246] present a Java RPC framework where small data is replicated and sent over a second TCP connection to the server or back to the client. At the destination, a listener collects all arriving data on all connections, reassembles the original data, and passes it to the corresponding handler.

Stuedi et al. [239] increase the efficiency of RPCs in data centers by softening the user land and kernel separation in the network stack and by using remote direct memory access to minimize the overhead of network operations by performing them with less context switches and zero-copy network I/O.

Chen et al. [49] introduce memory regions where server and client exchange data to improve the efficiency of RPCs between virtual machines (VMs) on the same host computer. The proposed framework has three components: (i) a notification channel that informs the server about new calls and the client about arriving results, (ii) a control channel that sends meta-data, and (iii) a transfer channel that is responsible for transmitting data between server and client and putting the data in the predefined memory regions.

Shyam et al. [228] propose solutions for situations where an RPC server is not available. The first solution is a heartbeat server that observes whether the RPC server is operative. The second solution is that every node sends a health check message to the RPC server. Since these messages are typically smaller than an RPC request and no computations take place, the answer of the health check should arrive faster. If the answer does not arrive within a timeout that is smaller than the timeout for the RPC, the server is considered inoperative.

Reinhardt et al. [199] address the problem of providing RPCs in wireless sensor networks. In particular, the authors eliminate the need of conventional RPCs to send predefined data to predefined destinations, typically addressed by ports, by publishing descriptions of new sensors that can be used by other sensors or nodes dynamically.

Shi et al. [223] present a framework where mobile devices can offload jobs to other mobile devices. In scenarios where node mobility is high, only small tasks will be offloaded; otherwise larger jobs will be offloaded, too. To increase the number of offloaded jobs, every job is split into smaller tasks. Additionally, every node has to announce its capabilities, such as CPU capacity and available battery power. To offload a job, the framework compares the task requirements with the capabilities of the client and tries to find a server that satisfies the requirements better than the client. If no server is found, the job will be executed locally.

Chen et al. [52] propose a solution for offloading computations to ad-hoc cloudlets. A job is offloaded via an ad-hoc communication channel that is closed after the procedure has been called successfully. The result of the job can arrive (i) via an ad-hoc channel if server and client are in proximity, (ii) via a cellular network used when an ad-hoc connection is not possible, (iii) via a Wi-Fi access point, if available.

Zhang et al. [275] propose a solution for cloudlets with intermittent connectivity where parts of a job will be executed either locally or remotely. The decision which of both options is chosen is based on a probability that includes the cost of executing a task. Two cost factors are calculated: (i) the cost when the phase is executed locally, where, e.g., energy consumption is important, (ii) the cost when the phase is executed in a cloudlet, where, e.g., available bandwidth is important. Based on this information, a Markov chain can be constructed, and the optimal path can be found.

Lai et al. [136] propose an offloading algorithm for delay-tolerant mobile networks that increases the amount of offloaded data without increasing the transmission overhead or delay. The transfer channel is chosen based on the contact duration between two nodes and the available transmission protocols. Therefore, every node logs which neighbors are available. Based on the available neighbors, on the size of the data that is offloaded, and the estimated

waiting time, a priority is computed. With these factors, a utility is calculated that denotes whether data should be offloaded using this particular channel or not.

To summarize, several of the related works address problems of RPCs in traditional networks, where links are either static or tasks are on the same machine, such as in VMs. Furthermore, direct memory access methods to reduce networking overhead cannot be used in a DTN environment, due to possibly untrustworthy nodes. Also, control mechanisms like heartbeats or duplicating data on multiple channels are no options for DTN. In the offloading approaches, the particular problems of RPCs in DTN are either not addressed or would require additional infrastructure, such as cell towers for 3G or LTE connectivity, or nodes with access to the Internet. The proposed DTN-RPC is designed to provide RPCs in DTN environments without requiring any additional infrastructure.

7.3.5 Summary

In this section, DTN-RPC was presented, a new approach to provide RPCs in device edge environments. DTN-RPC uses the concept of transitions to significantly increase the rate of successful RPCs. Furthermore, to decide which transport protocol to use for RPCs, i.e., to implement transparent transitions, both client and server comprehend their situation with respect to their network state based on perceived information like available mobile devices using network lookups so that it can be decided which transport protocol will be used for the call or the transmission of the result. Besides the use of using transitions to cope with challenging networks in device edge computing environments, DTN-RPC also differentiates between control and data channels to cope with potentially short contact durations in DTN where it is potentially impossible to transmit large amounts of data and uses explicit and implicit modes to address remote servers. Finally, servers use a situation-aware approach where information from their sensors but also predicates from clients are used to comprehend their situation based on their ability to execute the called procedure. The experimental results have indicated that the measured CPU and network overheads for DTN-RPC are reasonably low, so that DTN-RPC can be executed on mobile devices, and that the round-trip times and the number of successful RPCs are highly satisfactory in device edge environments. Thus, DTN-RPC adds computing capabilities in the form of RPCs to device edge environments.

There are several areas for future work. First, DTN-RPC has been tested and evaluated using emulated networks. To get a better view on the real-world performance of DTN-RPC, real mobile devices should be used. Second, since the Non-DTN transport protocols produced satisfactorily results in the *Chained* and *Hub* topologies, DTN-RPC should be evaluated without relying on the strict differentiation between control and data channels. Finally, although it is relatively difficult to implement error handling and acknowledgement mechanisms, our evaluation has shown that this is not impossible. Thus, an acknowledgement system should be implemented for the *any* mode to inform other servers that the execution has already started to reduce unnecessary computations.

7.4 Summary

This chapter discussed two case studies for situation-aware edge computing. They emphasize the feasibility, applicability, and importance of the novel concept of situation-aware edge computing. It was shown how situation-ware edge computing can contribute to emergency response applications and how to implement transitions in the area of device edge computing.

In particular, the following contributions were presented:

- *Situation-aware edge computing in emergency response applications:* Providing three applications in the areas of infrastructure edge computing, device edge computing, and embedded edge computing to support rescue helpers and affected people.
- *Situation-awareness for transitions at the edge:* Using information regarding the network status to comprehend a situation with respect to which transport protocol should be used and to transition between different transport protocols.

8 Conclusion

This chapter concludes this thesis and outlines areas of future work.

8.1 Summary

In this thesis, the novel approach of situation-aware edge computing was presented. It is based on the concepts of situation and situation-awareness, i.e., approaches that allow decision-making based on the perceived environmental information, comprehended to gather a domain-specific understanding of the environment.

Situation-aware edge computing was divided into three areas: (i) situation-aware infrastructure edge computing, (ii) situation-aware device edge computing, and (iii) situation-aware embedded edge computing.

For (i), the main research question was how to enable stakeholders of the Internet to make economic decisions using their current situations. In particular, the following two approaches were presented:

- A novel iterative bargaining approach between two stakeholders for nearly optimal service placement in infrastructure edge computing scenarios with respect to social cost despite incomplete information was proposed.
- A case study based on the mobile augmented reality game Ingress highlighted metrics and measures for situation-aware mobile augmented reality applications, where the relationship between the service provider's situation and the number of service usages was investigated.

For (ii), situation-aware offloading and connection loss prediction was realized:

- A novel situation-aware framework was presented for offloading computational workflows in opportunistic networks, where so-called workers announce their capabilities and available resources that are perceived by clients, i.e., the offloading devices, and comprehended to get their situations. Clients then can project their future situation to decide on which worker a task should be offloaded to.
- A novel situation-aware approach was presented to predict Wi-Fi connection loss to perform seamless vertical Wi-Fi/cellular handovers based on perceived sensor information on mobile devices; it predicts connection losses 15 seconds ahead using a machine learning approach.

For (iii), two approaches that enable situation-aware applications in the area of embedded edge computing were presented:

- With ReactiFi, a domain-specific language following the reactive programming paradigm, programmability of embedded devices was facilitated, where programmers use a high-level reactive programming language to perceive environmental information, comprehend the device's situation and make decisions based on the projection of future situations.
- Multimodal CEP is a novel approach to process event streams on embedded devices. In multimodal CEP, queries are formulated in a high-level language, which are broken up. The most adequate execution mode for the involved CEP operators is selected.

Finally, situation-aware edge computing was applied in two case studies:

- The novel idea of situation-aware edge computing for emergency response applications was introduced. By applying the concept of situation-awareness to emergency response applications, rescue helpers and people affected by emergencies are provided with useful information and applications that help to handle emergencies.
- A new approach to provide RPCs for device edge computing was presented that uses the concept of transitions to significantly increase the rate of successful RPCs.

8.2 Future Work

This section presents future work for situation-aware edge computing, which can be split into three parts: (i) situation-awareness, (ii) situation-aware infrastructure edge computing, situation-aware device edge computing, and situation-aware embedded edge computing, and (iii) putting it all together.

8.2.1 Situation-awareness

While the concept of situation-awareness has been explored in the field of cognitive science for several years, it is still quite new in computer science, which calls for further research. This thesis provides a unified definition of the concepts of situation and situation-awareness in computer science. However, this definition is exemplary and not formalized. Such formalization is necessary, however, for scientists to conduct research in situational-awareness on a common basis. Furthermore, in addition to a formal model, a common framework needs to be developed to enable systematic modeling and implementation of situation-aware systems. Such models can rely on research from context(-awareness) modeling, where a variety of approaches already exist, e.g., ontologies or system context diagrams for visualization.

8.2.2 Areas of Situation-aware Edge Computing

In the area of **situation-aware infrastructure edge computing**, infrastructure devices have constrained resources with respect to how many services a single infrastructure device can serve. This factor should be taken into account when designing a situation-aware placement approach. In addition, the aspect of incomplete information should be investigated further, especially when the information sources not only provide incomplete information, but the information

sources are adversaries and deliberately provide false information. Also, mechanisms to enforce truthfulness between stakeholders, such as reputation-based methods, should be investigated. Furthermore, the Internet consists of more than two stakeholders. The cost model should also consider cloud providers and users of the service. With respect to the analysis of environmental factors influencing the service provider's situation a broader analysis should be conducted. For example, the release of services from competitors has a significant influence. Additionally, this approach should be applied to other service types beyond mobile augmented reality games and adjusted accordingly to cope with specific features of different services.

In the area of **situation-aware device edge computing**, incorporating further network or social information could improve comprehension and projection further with respect to the decision where to offload a process to. Further, one promising network paradigm is Named Data Networking (NDN) [274]. Applying the concept of situation-awareness could be interesting, since clients or intermediate workers would not need to do the worker assignment, but use the abstraction NDN introduces to achieve the same results. Beyond deciding where to offload a process to, predicting connection loss also has interesting areas for future work. While the sensor information used for the approach presented in this thesis support high-quality predictions, other more domain-specific information might be useful to predict, e.g., Wi-Fi overloads. To use these approaches efficiently on off-the-shelf mobile devices, lightweight neural networks on dedicated processing engines should be considered.

In the area of **situation-aware embedded edge computing**, supporting an execution environment like eBPF could help to create secure platforms, where unknown code could be executed with limited security implications, allowing platforms for distributing and sharing arbitrary programs written in a reactive language. On the language side, the proposed DSL could support federated execution, allowing to program parts of an application on multiple embedded devices. Another interesting area is to extend the type system with assume-guarantee reasoning, by letting the programmer provide high-level specifications of the memory and real-time characteristics of the user-defined functions. This could then be used to harden the guarantees of the entire application. Regarding multimodal CEP, (i) changing the execution environment for operators to an interpreted bytecode approach to simplify compilation and deployment, (ii) introducing an approach to distribute operators to remote devices and to receive events via the network, and (iii) providing further operators, e.g., a pattern matcher or probabilistic CEP operators [255] would be interesting to investigate.

8.2.3 Situation-aware Edge Computing

This thesis introduced the novel concept of situation-aware edge computing by discussing the three individual areas of situation-aware infrastructure, device, and embedded edge computing. However, situation-aware applications utilizing all three of these areas into one system could even further improve qualities of existing services or enable novel applications. Consider, for example, a situation-aware edge computing system in the area of mobile AR games. Here, the service provider has to place the service on edge resources despite incomplete information as to where the users will play the game. Furthermore, the application running on users' devices executing the game will connect to the next available game instance, no matter whether it is deployed on a cloudlet or cloud. This system could be improved by utilizing the embedded systems on the users' devices for perceiving the users' context information in an

energy-efficient manner similar to the proposed approaches throughout this thesis. Based on the perceived information from embedded systems as well as further perceived information from sources not usable with embedded devices (e.g., user preferences), the users' situations regarding their application usage behavior can then be comprehended directly on a device utilizing situation-aware device edge concepts. This on-device situation comprehension could then be used by service providers to predict where users are going to play their game and deploy the service on cloudlets where users will play the game.

Such a system requires multimodal analysis models to provide situation-awareness across all three edge computing areas simultaneously. One of the biggest challenges is the fact that the three edge computing areas are heterogeneous in terms of computing power. Thus, a way must be found to enable the processing of a wide variety of data types in a variety of computing environments, from embedded systems to servers. Furthermore, if this is not possible to process data locally, for example because the data rate exceeds the capacity of an embedded system, communication methods are needed that enable efficient and fast communication between the three areas of edge computing. Maintaining high efficiency in such protocols is important to not negate the energy-saving advantages of embedded systems, but also of mobile devices such as smartphones. High execution speed requirements must be met to not undermine the low latency advantages of edge computing in general. Finally, such a system also needs a feedback channel that enables a decision made on the basis of a projection to be converted into a corresponding action, and the resulting context changes must be taken into account.

List of Figures

List of Tables

Bibliography

[1] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. "Mobile edge computing: A survey." In: *IEEE Internet of Things Journal* 5.1. 2017, pp. 450–465 (pages 28, 87).

[2] Saeid Abolfazli, Zohreh Sanaei, Ejaz Ahmed, Abdullah Gani, and Rajkumar Buyya. "Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges." In: *IEEE Communications Surveys & Tutorials* 16.1. 2013, pp. 337–368 (page 27).

[3] 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: *International Symposium on Handheld and Ubiquitous Computing*. Springer. 1999, pp. 304–307 (page 34).

[4] Brett Adams, Dinh Phung, and Svetha Venkatesh. "Sensing and using social context." In: *ACM Transactions on Multimedia Computing, Communications, and Applications* 5.2. 2008, pp. 1–27 (page 77).

[5] Atiq Ahmed, Leila Merghem Boulahia, and Dominique Gaiti. "Enabling vertical handover decisions in heterogeneous wireless networks: A state-of-the-art and a classification." In: *IEEE Communications Surveys & Tutorials* 16.2. 2013, pp. 776–811 (page 115).

[6] Zaafar Ahmed, Muhammad Hamad Alizai, and Affan A Syed. "Inkev: In-kernel distributed network virtualization for DCN." In: *ACM SIGCOMM Computer Communication Review* 46.3. 2018, pp. 1–6 (page 161).

[7] Sanghong Ahn, Joohyung Lee, Sangdon Park, SH Shah Newaz, and Jun Kyun Choi. "Competitive partial computation offloading for maximizing energy efficiency in mobile cloud computing." In: *IEEE Access* 6. 2018, pp. 899–912 (page 100).

[8] Jeff Ahrenholz, Claudiu Danilov, Thomas R Henderson, and Jae H Kim. "CORE: A real-time network emulator." In: *IEEE Military Communications Conference*. IEEE. 2008, pp. 1–7 (page 96).

[9] Adnan Akbar, Francois Carrez, Klaus Moessner, Juan Sancho, and Juan Rico. "Context-aware stream processing for distributed IoT applications." In: *IEEE 2nd World Forum on Internet of Things*. IEEE. 2015, pp. 663–668 (page 161).

[10] Bastian Alt, Markus Weckesser, Christian Becker, Matthias Hollick, Sounak Kar, Anja Klein, Robin Klose, Roland Kluge, Heinz Koeppl, Boris Koldehofe, et al. "Transitions: A protocol-independent view of the future Internet." In: *Proceedings of the IEEE* 107.4. 2019, pp. 835–846 (page 29).

[11] Sven Amann, Hoan Anh Nguyen, Sarah Nadi, Tien N. Nguyen, and Mira Mezini. "A systematic evaluation of static API-misuse detectors." In: *IEEE Transactions on Software Engineering* 45.12. 2018, pp. 1170–1188 (page 132).

[12] Amazon.com, Inc. "Amazon AWS Lambda". 2021. URL: `https://aws.amazon.com/lambda/` (visited on 07/01/2021) (pages 58, 64, 65, 67).

[13] Amazon.com, Inc. "Amazon AWS Lambda Edge". 2021. URL: `https://aws.amazon.com/lambda/edge/` (visited on 07/01/2021) (pages 58, 64, 65, 67).

[14] Ogbonnaya Anicho, Philip B Charlesworth, Gurvinder S Baicher, and Atulya Nagar. "Situation awareness and routing challenges in unmanned HAPS/UAV based communications networks." In: *International Conference on Unmanned Aircraft Systems*. IEEE. 2020, pp. 1175–1182 (page 36).

[15] Aleksandar Antonic, Kristijan Roankovic, Martina Marjanovic, Kreimir Pripuic, and Ivana Podnar Zarko. "A mobile crowdsensing ecosystem enabled by a cloud-based publish/subscribe middleware." In: *International Conference on Future Internet of Things and Cloud*. IEEE. 2014, pp. 107–114 (page 161).

[16] Arvind Arasu, Shivnath Babu, and Jennifer Widom. "The CQL continuous query language: Semantic foundations and query execution." In: *The VLDB Journal* 15.2. 2006, pp. 121–142 (page 147).

[17] "Augmented reality games: How much do they really cost". 2020 (pages 65, 66).

[18] Emmanuel Baccelli, Oliver Hahm, Mesut Gunes, Matthias Wahlisch, and Thomas C. Schmidt. "RIOT OS: Towards an OS for the Internet of Things." In: *IEEE Conference on Computer Communications Workshops*. Turin, Italy: IEEE, 2013, pp. 79–80 (page 137).

[19] Engineer Bainomugisha, Andoni Lombide Carreton, Tom van Cutsem, Stijn Mostinckx, and Wolfgang de Meuter. "A survey on reactive programming." In: *ACM Computing Surveys* 45.4. 2013, pp. 1–34 (pages 21–23).

[20] Saleh Basalamah, Sultan Daud Khan, and Habib Ullah. "Scale driven convolutional neural network model for people counting and localization in crowd scenes." In: *IEEE Access*. 2019 (page 172).

[21] Lars Baumgärtner, Paul Gardner-Stephen, Pablo Graubner, Jeremy Lakeman, Jonas Höchst, Patrick Lampe, Nils Schmidt, Stefan Schulz, Artur Sterz, and Bernd Freisleben. "An experimental evaluation of delay-tolerant networking with Serval." In: *IEEE Global Humanitarian Technology Conference*. IEEE. 2016, pp. 70–79 (pages 92, 94, 166, 178, 182, 184, 186).

[22] Lars Baumgärtner, Pablo Graubner, Jonas Höchst, Anja Klein, and Bernd Freisleben. "Speak less, hear enough: On dynamic announcement intervals in wireless on-demand networks." In: *13th Annual Conference on Wireless On-demand Network Systems & Services*. IEEE. 2017, pp. 33–40 (page 96).

[23] Lars Baumgärtner, Jonas Höchst, Patrick Lampe, Ragnar Mogk, Artur Sterz, Pascal Weisenburger, Mira Mezini, and Bernd Freisleben. "Smart street lights and mobile citizen apps for resilient communication in a digital city." In: *IEEE Global Humanitarian Technology Conference*. IEEE. 2019, pp. 1–8 (pages xiv, 165, 166).

[24] Lars Baumgärtner, Alvar Penning, Patrick Lampe, Björn Richerzhagen, Ralf Steinmetz, and Bernd Freisleben. "Environmental monitoring using low-cost hardware and infrastructureless wireless communication." In: *IEEE Global Humanitarian Technology Conference*. IEEE. 2018, pp. 1–8 (page 89).

[25] Paolo Bellavista, Antonio Corradi, Mario Fanelli, and Luca Foschini. "A survey of context data distribution for mobile ubiquitous systems." In: *ACM Computing Surveys* 44.4. 2012, pp. 1–45 (pages 76, 146).

[26] Paolo Bellavista, Alessandro Zanni, and Michele Solimando. "A migration-enhanced edge computing support for mobile devices in hostile environments." In: *13th International Wireless Communications and Mobile Computing Conference*. IEEE. 2017, pp. 957–962 (page 101).

[27] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing." In: *Future Generation Computer Systems* 28.5. 2012, pp. 755–768 (page 86).

[28] Carlos J. Bernardos, Antonio De La Oliva, Pablo Serrano, Albert Banchs, Luis M. Contreras, Hao Jin, and Juan Carlos Zúñiga. "An architecture for software defined wireless networking." In: *IEEE Wireless Communications* 21.3. 2014, pp. 52–61 (page 137).

[29] Gérard Berry and Georges Gonthier. "The Esterel synchronous programming language: Design, semantics, implementation." In: *Science of Computer Programming* 19.2. 1992, pp. 87–152 (page 138).

[30] Dimitri P Bertsekas, Robert G Gallager, and Pierre Humblet. "Data networks". Vol. 2. Prentice-Hall International New Jersey, 1992 (page 70).

[31] Giuseppe Bianchi, Pierluigi Gallo, Domenico Garlisi, Fabrizio Giuliano, Francesco Gringoli, and Ilenia Tinnirello. "MAClets: Active MAC protocols over hard-coded devices." In: *8th International Conference on Emerging Networking Experiments and Technologies*. Nice, France: ACM, 2012, pp. 229–240 (page 137).

[32] Andrew D Birrell and Bruce Jay Nelson. "Implementing remote procedure calls." In: *ACM Transactions on Computer Systems* 2.1. 1984, pp. 39–59 (page 177).

[33] Chiara Boldrini, Kyunghan Lee, Melek Önen, Jörg Ott, and Elena Pagani. "Opportunistic networks." In: *Computer Communications* 48.14. 2014, pp. 1–4 (page 12).

[34] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. "P4: Programming protocol-independent packet processors." In: *ACM SIGCOMM Computer Communication Review* 44.3. 2014, pp. 87–95 (page 137).

[35] Irina Botan, Younggoo Cho, Roozbeh Derakhshan, Nihal Dindar, Laura M Haas, Kihong Kim, Chulwon Lee, Girish Mundada, Ming-Chien Shan, Nesime Tatbul, et al. "Design and implementation of the MaxStream federated stream processing architecture." In: *Technical Report/ETH Zurich, Department of Computer Science* 632. 2009 (page 161).

[36] Lawrence Brakmo. "TCP-BPF: Programmatically tuning TCP behavior through BPF." In: *2.2 Technical Confernence on Linux Networking*. Netdev. 2017, pp. 1–5 (page 136).

[37] Eyuphan Bulut and Boleslaw K Szymanski. "Rethinking offloading Wi-Fi access point deployment from user perspective." In: *IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications*. IEEE. 2016, pp. 1–6 (page 87).

[38] Eyuphan Bulut and Boleslaw K Szymanski. "Wi-Fi access point deployment for efficient mobile data offloading." In: *ACM SIGMOBILE Mobile Computing and Communications Review* 17.1. 2013, pp. 71–78 (page 87).

[39] Scott Burleigh. "Minimal TCP convergence-layer protocol." In: *Working Draft, IETF Secretariat, Internet-Draft draft-ietf-dtn-mtcpcl-01*. 2019 (page 11).

[40] Scott Burleigh, Kevin Fall, and Edward Birrane. "Bundle Protocol Version 7". RFC 9171. RFC Editor, 2022 (pages 6, 11, 12).

[41] Jian Cao, Jiwen Fu, Minglu Li, and Jinjun Chen. "CPU load prediction for cloud environment based on a dynamic ensemble model." In: *Software: Practice and Experience* 44.7. 2014, pp. 793–804 (pages 85, 86).

[42] Xiaofeng Cao, Guoming Tang, Deke Guo, Yan Li, and Weiming Zhang. "Edge federation: Towards an integrated service provisioning model." In: *IEEE/ACM Transactions on Networking* 28.3. 2020, pp. 1116–1129 (pages 72, 73).

[43] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. "Ethane: Taking control of the enterprise." In: *ACM SIGCOMM Computer Communication Review* 37.4. 2007, pp. 1–12 (page 137).

[44] Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman, Dan Boneh, Nick McKeown, and Scott Shenker. "SANE: A protection architecture for enterprise networks." In: *USENIX Security Symposium*. Vol. 49. 2006, p. 50 (page 137).

[45] Vinton Cerf. "Delay-tolerant network architecture: The evolving interplanetary Internet". Internet-Draft draft-irtf-ipnrg-arch-01. `https://www.ietf.org/archive/id/draft-irtf-ipnrg-arch-01.txt`. IETF Secretariat, 2002 (page 11).

[46] Vinton Cerf, Scott Burleigh, Adrian Hooke, Leigh Torgerson, Robert Durst, Keith Scott, Kevin Fall, and Howard Weiss. "Delay-tolerant networking architecture." In: 2007 (page 11).

[47] Vikram Chandrasekhar, Jeffrey G Andrews, and Alan Gatherer. "Femtocell networks: A survey." In: *IEEE Communications Magazine* 46.9. 2008, pp. 59–67 (page 87).

[48] Dimitris Chatzopoulos, Mahdieh Ahmadi, Sokol Kosta, and Pan Hui. "Have you asked your neighbors? A Hidden Market approach for device-to-device offloading." In: *17th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks*. IEEE. 2016, pp. 1–9 (page 101).

[49] Hao Chen, Lin Shi, Jianhua Sun, Kenli Li, and Ligang He. "A fast RPC system for virtual machines." In: *IEEE Transactions on Parallel and Distributed Systems* 24.7. 2012, pp. 1267–1276 (page 190).

[50] Lixing Chen, Cong Shen, Pan Zhou, and Jie Xu. "Collaborative service placement for edge computing in dense small cell networks." In: *IEEE Transactions on Mobile Computing*. 2019 (pages 72, 73).

[51] Lixing Chen, Jie Xu, Shaolei Ren, and Pan Zhou. "Spatio–temporal edge service placement: A bandit learning approach." In: *IEEE Transactions on Wireless Communications* 17.12. 2018, pp. 8388–8401 (pages 52, 72).

[52] Min Chen, Yixue Hao, Yong Li, Chin-Feng Lai, and Di Wu. "On the computation offloading at ad hoc cloudlet: Architecture and service modes." In: *IEEE Communications Magazine* 53.6. 2015, pp. 18–24 (page 190).

[53] 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* 5. 2016, pp. 2795–2808 (page 101).

[54] Yung-Chih Chen, Yeon-sup Lim, Richard J Gibbens, Erich M Nahum, Ramin Khalili, and Don Towsley. "A measurement-based study of multipath tcp performance over wireless networks." In: *Conference on Internet Measurement*. 2013, pp. 455–468 (page 116).

[55] Kefei Cheng and Yanglei Cui. "Design and implementation of network packets collection tools based on the android platform." In: *9th International Conference on Fuzzy Systems and Knowledge Discovery*. IEEE. 2012, pp. 2166–2169 (page 161).

[56] Sung In Cho and Suk-Ju Kang. "Real-time people counting system for customer movement analysis." In: *IEEE Access* 6. 2018, pp. 55264–55272 (page 172).

[57] Pushpinder Kaur Chouhan, Sally McClean, and Mark Shackleton. "Situation assessment to secure IoT applications." In: *5th International Conference on Internet of Things: Systems, Management and Security*. IEEE. 2018, pp. 70–77 (page 36).

[58] Mahfuzulhoq Chowdhury, Eckehard Steinbach, Wolfgang Kellerer, and Martin Maier. "Context-aware task migration for HART-Centric collaboration over Wi-Fi based tactile Internet infrastructures." In: *IEEE Transactions on Parallel and Distributed Systems* 29.6. 2018, pp. 1231–1246 (page 101).

[59] Thomas Clausen, Philippe Jacquet, Cédric Adjih, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum, and Laurent Viennot. "Optimized link state routing protocol (OLSR)." In: 2003 (page 182).

[60] IEEE Computer Society LAN/MAN Standards Committee et al. "IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications." In: *IEEE Std 802.11^*. 2007 (page 15).

[61] Quentin De Coninck, Matthieu Baerts, Benjamin Hesmans, and Olivier Bonaventure. "A first analysis of multipath tcp on smartphones." In: *International Conference on Passive and Active Network Measurement*. Springer. 2016, pp. 57–69 (page 110).

[62] Marco Conti, Silvia Giordano, Martin May, and Andrea Passarella. "From opportunistic networks to opportunistic computing." In: *IEEE Communications Magazine* 48.9. 2010, pp. 126–139 (page 89).

[63] Gregory H. Cooper and Shriram Krishnamurthi. "Embedding dynamic dataflow in a call-by-value language." In: *European Symposium on Programming*. Vienna, Austria: Springer, 2006, pp. 294–308 (page 121).

[64] Salvatore Costanzo, Laura Galluccio, Giacomo Morabito, and Sergio Palazzo. "Software defined wireless networks: Unbridling SDNs." In: *European Workshop on Software Defined Networking*. Darmstadt, Germany: IEEE, 2012, pp. 1–6 (page 137).

[65] Quentin De Coninck and Olivier Bonaventure. "Every millisecond counts: Tuning Multipath TCP for interactive applications on smartphones." In: *Technical report*. 2017 (page 116).

[66] Quentin De Coninck, François Michel, Maxime Piraux, Florentin Rochet, Thomas Given-Wilson, Axel Legay, Olivier Pereira, and Olivier Bonaventure. "Pluginizing QUIC." In: *ACM Special Interest Group on Data Communication*. 2019, pp. 59–74 (page 136).

[67] Michael Demmer and Kevin Fall. "DTLSR: Delay tolerant routing for developing regions." In: *Workshop on Networked Systems for Developing Regions*. 2007, pp. 1–6 (page 11).

[68] Shuiguang Deng, Longtao Huang, Javid Taheri, and Albert Y Zomaya. "Computation offloading for service workflow in mobile cloud computing." In: *IEEE Transactions on Parallel and Distributed Systems* 26.12. 2015, pp. 3317–3329 (page 101).

[69] Thinh Quang Dinh, Quang Duy La, Tony QS Quek, and Hyundong Shin. "Learning for computation offloading in mobile edge computing." In: *IEEE Transactions on Communications* 66.12. 2018, pp. 6353–6367 (pages 65, 66).

[70] Martin Duggan, Karl Mason, Jim Duggan, Enda Howley, and Enda Barrett. "Predicting host CPU utilization in cloud computing using recurrent neural networks." In: *12th International Conference for Internet Technology and Secured Transactions*. IEEE. 2017, pp. 67–72 (page 85).

[71] Francis T Durso, Raymond S Nickerson, Susan T Dumais, Stephan Lewandowsky, and Timothy J Perfect. "Handbook of applied cognition". John Wiley & Sons, 2007 (page 36).

[72] Jonathan Edwards. "Coherent reaction." In: *24th ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*. Orlando, FL, USA: ACM, 2009, pp. 925–932 (page 119).

[73] Hanan H Elazhary and Sahar F Sabbeh. "The W 5 framework for Computation offloading in the Internet of Things." In: *IEEE Access* 6. 2018, pp. 23883–23895 (page 102).

[74] Conal Elliott and Paul Hudak. "Functional reactive animation." In: *2nd ACM SIGPLAN International Conference on Functional Programming*. 1997, pp. 263–273 (page 21).

[75] Mica R Endsley. "Design and evaluation for situation-awareness enhancement." In: *Human Factors Society Annual Meeting*. Vol. 32. 2. Sage Publications Sage CA: Los Angeles, CA. 1988, pp. 97–101 (page 36).

[76] Mica R Endsley. "Toward a theory of situation-awareness in dynamic systems." In: *Human Factors* 37.1. 1995, pp. 32–64 (page 33).

[77] Kevin Fall. "A delay-tolerant network architecture for challenged Internets." In: *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 2003, pp. 27–34 (pages 11, 177).

[78] Wenhao Fan, Yuan'an Liu, Bihua Tang, Fan Wu, and Zhongbao Wang. "Computation offloading based on cooperations of mobile edge computing-enabled base stations." In: *IEEE Access* 6. 2018, pp. 22622–22633 (page 101).

[79] Farhad Farokhi and Iman Shames. "Bilateral trade under information asymmetry and quantized measurements." In: *IEEE Conference on Decision and Control*. IEEE. 2018, pp. 5832–5837 (page 62).

[80] Patrick Felka, Artur Sterz, Oliver Hinz, and Bernd Freisleben. "Using social media to estimate the audience sizes of public events for crisis management and emergency care." In: *International Conference on Smart Health*. Springer. 2018, pp. 77–89 (page 166).

[81] 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: *17th International Conference on Mobile and Ubiquitous Multimedia*. 2018, pp. 25–35 (pages xiii, 47, 58, 63, 65, 74).

[82] Jie Feng, Liqiang Zhao, Jianbo Du, Xiaoli Chu, and F Richard Yu. "Computation offloading and resource allocation in D2D-enabled mobile edge computing." In: *IEEE International Conference on Communications*. IEEE. 2018, pp. 1–6 (page 102).

[83] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. "Computing with nearby mobile devices: A work sharing algorithm for mobile edge-clouds." In: *IEEE Transactions on Cloud Computing* 1. 2016, pp. 1–1 (page 101).

[84] Spencer P. Florence, Shu-Hung You, Jesse A. Tov, and Robert Bruce Findler. "A calculus for Esterel: If can, can. If no can, no can." In: *ACM on Programming Languages* 3.POPL. 2019, pp. 1–29 (page 138).

[85] Huber Flores and Satish Srirama. "Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning." In: *4th ACM Workshop on Mobile Cloud Computing and Services*. ACM. 2013, pp. 9–16 (page 101).

[86] Ramon Dos Reis Fontes, Claudia Campolo, Christian Esteve Rothenberg, and Antonella Molinaro. "From theory to experimental evaluation: Resource management in software-defined vehicular networks." In: *IEEE Access* 5. 2017, pp. 3069–3076 (page 137).

[87] Alan Ford, Costin Raiciu, Mark Handley, Olivier Bonaventure, et al. "TCP extensions for multipath operation with multiple addresses." In: 2013 (page 116).

[88] Nate Foster, Arjun Guha, Mark Reitblatt, Alec Story, Michael J Freedman, Naga Praveen Katta, Christopher Monsanto, Joshua Reich, Jennifer Rexford, Cole Schlesinger, et al. "Languages for software-defined networks." In: *IEEE Communications Magazine* 51.2. 2013, pp. 128–134 (page 137).

[89] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. "Frenetic: A network programming language." In: vol. 46. 9. ACM, 2011, pp. 279–291 (page 138).

[90] Alexander Frömmgen, Mohamed Hassan, Roland Kluge, Mahdi Mousavi, Max Mühlhäuser, Sabrina Müller, Mathias Schnee, Michael Stein, and Markus Weckesser. "Mechanism transitions: A new paradigm for a highly adaptive Internet." In: 2016 (page 29).

[91] Alexander Frömmgen, Amr Rizk, Tobias Erbshäußer, Max Weller, Boris Koldehofe, Alejandro Buchmann, and Ralf Steinmetz. "A programming model for application-defined multipath TCP scheduling." In: *18th ACM/IFIP/USENIX Middleware Conference*. ACM. 2017, pp. 134–146 (page 136).

[92] Colin Funai, Cristiano Tapparello, and Wendi Heinzelman. "Mobile to mobile computational offloading in multi-hop cooperative networks." In: *IEEE Global Communications Conference*. IEEE. 2016, pp. 1–7 (page 101).

[93] Lin Gao, George Iosifidis, Jianwei Huang, Leandros Tassiulas, and Duozhe Li. "Bargaining-based mobile data offloading." In: *IEEE Journal on Selected Areas in Communications* 32.6. 2014, pp. 1114–1125 (page 54).

[94] Paul Gardner-Stephen, Andrew Bettison, Romana Challans, and Jeremy Lakeman. "The rational behind the serval network layer for resilient communications." In: *Journal of Computer Science* 9.12. 2013, p. 1680 (pages 178, 182).

[95] Paul Gardner-Stephen, Romana Challans, Jeremy Lakeman, Andrew Bettison, Dione Gardner-Stephen, and Matthew Lloyd. "The serval mesh: A platform for resilient communications in disaster & crisis." In: *IEEE Global Humanitarian Technology Conference*. IEEE. 2013, pp. 162–166 (pages 178, 182).

[96] Paul Gardner-Stephen, Jeremy Lakeman, Romana Challans, Corey Wallis, Ariel Stulman, and Yoram Haddad. "MeshMS: Ad hoc data transfer within a mesh network." In: *International Journal of Communications, Network and System Sciences* 8.5. 2012, pp. 496–504 (pages 178, 182).

[97] Paul Gardner-Stephen and Swapna Palaniswamy. "Serval mesh software-Wi-Fi multi model management." In: *1st International Conference on Wireless Technologies for Humanitarian Relief*. ACM. 2011, pp. 71–77 (pages 6, 94, 177).

[98] David Gay, Philip Levis, Robert Von Behren, Matt Welsh, Eric Brewer, and David Culler. "The nesC language: A holistic approach to networked embedded systems." In: *ACM SIGPLAN Notices* 38.5. 2003, pp. 1–11 (page 137).

[99] 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: *IEEE/ACM Symposium on Edge Computing*. IEEE. 2018, pp. 187–202 (page 72).

[100] Google. "Cloud Functions — Google Cloud". 2021 (pages 58, 64, 65, 67).

[101] Mark S Gordon, Davoud Anoushe Jamshidi, Scott A Mahlke, Zhuoqing Morley Mao, and Xu Chen. "COMET: Code offload by migrating execution transparently." In: *OSDI*. Vol. 12. 2012, pp. 93–106 (page 100).

[102] Pablo Graubner, Patrick Lampe, Jonas Höchst, Lars Baumgärtner, Mira Mezini, and Bernd Freisleben. "Opportunistic named functions in disruption-tolerant emergency networks." In: *15th ACM International Conference on Computing Frontiers*. ACM. 2018, pp. 129–137 (page 97).

[103] Pablo Graubner, Matthias Schmidt, and Bernd Freisleben. "Energy-efficient virtual machine consolidation." In: *It Professional* 15.2. 2012, pp. 28–34 (page 86).

[104] Pablo Graubner, Christoph Thelen, Michael Körber, Artur Sterz, Guido Salvaneschi, Mira Mezini, Bernhard Seeger, and Bernd Freisleben. "Multimodal complex event processing on mobile devices." In: *12th ACM International Conference on Distributed and Event-Based Systems*. 2018, pp. 112–123 (pages xiii, xiv, 119, 145, 165, 166).

[105] Pablo Karl Graubner. "Energy-efficient transitional near-* computing." PhD thesis. University of Marburg, Germany, 2019 (pages 29, 30).

[106] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. "A clean slate 4D approach to network control and management." In: *ACM SIGCOMM Computer Communication Review* 35.5. 2005, pp. 41–54 (page 137).

[107] Muhammad Shams Ul Haq, Lejian Liao, and Ma Lerong. "Design and implementation of sandbox technique for isolated applications." In: *IEEE Information Technology, Networking, Electronic and Automation Control Conference*. IEEE. 2016, pp. 557–561 (page 161).

[108] Seppo Hätönen, Petri Savolainen, Ashwin Rao, Hannu Flinck, and Sasu Tarkoma. "Off-the-shelf software-defined Wi-Fi networks." In: *ACM SIGCOMM Conference*. Florianópolis, Brazil: ACM, 2016, pp. 609–610 (page 137).

[109] Caleb Helbling and Samuel Z. Guyer. "Juniper: A functional reactive programming language for the Arduino." In: *4th International Workshop on Functional Art, Music, Modelling, and Design*. 2016, pp. 8–16 (page 137).

[110] Oliver Hinz, Shawndra Hill, and Ju-Young Kim. "TV's dirty little secret: The negative effect of popular TV on online auction sales." In: *MIS Quarterly* 40.3. 2016, pp. 623–644 (page 86).

[111] Jonas Höchst, Lars Baumgärtner, Franz Kuntke, Alvar Penning, Artur Sterz, and Bernd Freisleben. "LoRa-based device-to-device smartphone communication for crisis scenarios." In: *17th International Conference on Information Systems for Crisis Response and Management*. ISCRAM. 2020, pp. 96–1011 (pages 6, 166).

[112] Jonas Höchst, Artur Sterz, Alexander Frömmgen, Denny Stohr, Ralf Steinmetz, and Bernd Freisleben. "Learning Wi-Fi connection loss predictions for seamless vertical handovers using multipath TCP." In: *IEEE 44th Conference on Local Computer Networks*. IEEE. 2019, pp. 18–25 (pages xiii, 6, 89, 103).

[113] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. "The express data path: Fast programmable packet processing in the operating system kernel." In: *14th International Conference on Emerging Networking Experiments and Technologies*. ACM. 2018, pp. 54–66 (page 136).

[114] Sana Horrich, Sana Ben Jamaa, and Philippe Godlewski. "Adaptive vertical mobility decision in heterogeneous networks." In: *3rd International Conference on Wireless and Mobile Communications*. IEEE. 2007, pp. 44–44 (page 115).

[115] SK Alamgir Hossain, Md Anisur Rahman, and M Anwar Hossain. "Edge computing framework for enabling situation-awareness in IoT based smart city." In: *Journal of Parallel and Distributed Computing* 122. 2018, pp. 226–237 (page 34).

[116] Chung-Ming Huang, Kun-chan Lan, and Chang-Zhou Tsai. "A survey of opportunistic networks." In: *22nd International Conference on Advanced Information Networking and Applications-Workshops*. IEEE. 2008, pp. 1672–1677 (page 12).

[117] Martin Husak, Tomas Jirsik, and Shanchieh Jay Yang. "SoK: Contemporary issues and challenges to enable cyber situational awareness for network security." In: *15th International Conference on Availability, Reliability and Security*. 2020, pp. 1–10 (page 36).

[118] International Telecommunication Union (ITU). "Information Technology - Open Systems Interconnection - Connectionless Presenation Protocol: Protocol Specification". Tech. rep. Telecommunication Standardization Sector of ITU, 1995 (page 10).

[119] Michaela Iorga, Larry Feldman, Robert Barton, Michael J Martin, Nedim S Goren, Charif Mahmoudi, et al. "Fog computing conceptual model." In: 2018 (pages 26, 27).

[120] Muhammad Azhar Iqbal, Sajjad Hussain, Huanlai Xing, and Muhammad Ali Imran. "Enabling the Internet of Things: Fundamentals, design and applications". John Wiley & Sons, 2020 (page 18).

[121] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. "Empirical prediction models for adaptive resource provisioning in the cloud." In: *Future Generation Computer Systems* 28.1. 2012, pp. 155–162 (pages 85, 86).

[122] Yashpalsinh Jadeja and Kirit Modi. "Cloud computing-concepts, architecture and challenges." In: *International Conference on Computing, Electronics and Electrical Technologies*. IEEE. 2012, pp. 877–880 (page 26).

[123] Mike Jia, Jiannong Cao, and Weifa Liang. "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks." In: *IEEE Transactions on Cloud Computing* 5.4. 2015, pp. 725–737 (pages 72, 73).

[124] David Johnson, Ntsibane S Ntlatlapa, and Corinna Aichele. "Simple pragmatic approach to mesh routing using BATMAN." In: *2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries*. Pretoria, South Africa, 2008 (page 182).

[125] Padmavathy Kankanala, Sanjoy Das, and Anil Pahwa. "AdaBoost: An ensemble learning approach for estimating weather-related outages in distribution systems." In: *IEEE Transactions on Power Systems* 29.1. 2013, pp. 359–367 (pages 85, 86).

[126] Marcin Kawecki and Radosław Olgierd Schoeneich. "Mobility-based routing algorithm in delay tolerant networks." In: *EURASIP Journal on Wireless Communications and Networking* 2016.1. 2016, pp. 1–9 (page 177).

[127] Shwet Ketu and Pramod Kumar Mishra. "Cloud, fog and mist Computing in IoT: An indication of emerging opportunities." In: *IETE Technical Review*. 2021, pp. 1–12 (page 119).

[128] Murad Khan, Awais Ahmad, Shehzad Khalid, Syed Hassan Ahmed, Sohail Jabbar, and Jamil Ahmad. "Fuzzy based multi-criteria vertical handover decision modeling in heterogeneous wireless networks." In: *Multimedia Tools and Applications* 76.23. 2017, pp. 24649–24674 (page 115).

[129] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. "Edge computing: A survey." In: *Future Generation Computer Systems* 97. 2019, pp. 219–235 (page 28).

[130] Dragi Kimovski, Roland Mathá, Josef Hammer, Narges Mehran, Hermann Hellwagner, and Radu Prodan. "Cloud, fog, or edge: Where to compute?" In: *IEEE Internet Computing* 25.4. 2021, pp. 30–36 (page 28).

[131] Mikkel Baun Kjærgaard, Jakob Langdal, Torben Godsk, and Thomas Toftkjær. "Entracked: Energy-efficient robust position tracking for mobile devices." In: *7th International Conference on Mobile Systems, Applications, and Services*. 2009, pp. 221–234 (page 161).

[132] Jürgen Krämer and Bernhard Seeger. "Semantics and implementation of continuous sliding window queries over data streams." In: *ACM Transactions on Database Systems* 34.1. 2009, pp. 1–49 (pages 145, 146).

[133] Yana Krytska, Inna Skarga-Bandurova, and Artem Velykzhanin. "IoT-based situation-awareness support system for real-time emergency management." In: *9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*. Vol. 2. IEEE. 2017, pp. 955–960 (page 36).

[134] Ian Ku, You Lu, Mario Gerla, Rafael L. Gomes, Francesco Ongaro, and Eduardo Cerqueira. "Towards software-defined VANETs: Architecture and services." In: *13th Annual Mediterranean ad hoc Networking Workshop)*. Piran, Slovenia: IEEE, 2014, pp. 103–110 (page 137).

[135] Jarno Lähteenmäki, Heikki Hämmäinen, Nan Zhang, and Matti Swan. "Cost modeling of a network service provider cloud platform." In: *IEEE International Conference on Cloud Engineering Workshop*. IEEE. 2016, pp. 148–153 (page 58).

[136] Yongxuan Lai, Xing Gao, Minghong Liao, Jinshan Xie, Ziyu Lin, and Haiying Zhang. "Data gathering and offloading in delay tolerant mobile networks." In: *Wireless Networks* 22.3. 2016, pp. 959–973 (page 190).

[137] Patrick Lampe, Lars Baumgärtner, Ralf Steinmetz, and Bernd Freisleben. "Smartface: Efficient face detection on smartphones for wireless on-demand emergency networks." In: *24th International Conference on Telecommunications*. IEEE. 2017, pp. 1–7 (pages 91, 96, 170).

[138] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. "The QUIC transport protocol: Design and Internet-scale deployment." In: *Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 183–196 (page 10).

[139] Edward Ashford Lee and Sanjit Arunkumar Seshia. "Introduction to embedded systems: A cyber-physical systems approach". MIT Press, 2016 (pages 17, 18).

[140] Matthew Lentz, James Litton, and Bobby Bhattacharjee. "Drowsy power management." In: *25th Symposium on Operating Systems Principles*. Monterey, California, 2015, pp. 230–244 (page 161).

[141] Philip Levis, Samuel Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, et al. "TinyOS: An operating system for sensor networks." In: *Ambient Intelligence*. Springer, 2005, pp. 115–148 (pages 137, 161).

[142] Yi Li, Hong Liu, Wenjun Yang, Dianming Hu, Xiaojing Wang, and Wei Xu. "Predicting inter-data-center network traffic using elephant flow and sublink information." In: *IEEE Transactions on Network and Service Management* 13.4. 2016, pp. 782–792 (pages 58, 85).

[143] Yuchen Li, Weifa Liang, and Jing Li. "Profit maximization for service placement and request assignment in edge computing via deep reinforcement learning." In: *24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. 2021, pp. 51–55 (page 67).

[144] Yu Liang, Jidong Ge, Sheng Zhang, Jie Wu, Lingwei Pan, Tengfei Zhang, and Bin Luo. "Interaction-oriented service entity placement in edge computing." In: *IEEE Transactions on Mobile Computing*. 2019 (pages 72, 73).

[145] Daniyal Liaqat, Silviu Jingoi, Eyal de Lara, Ashvin Goel, Wilson To, Kevin Lee, Italo De Moraes Garcia, and Manuel Saldana. "Sidewinder: An energy efficient and developer friendly heterogeneous architecture for continuous mobile sensing." In: *ACM SIGPLAN Notices* 51.4. 2016, pp. 205–215 (page 161).

[146] Andy Liaw, Matthew Wiener, et al. "Classification and regression by RandomForest." In: *R News* 2.3. 2002, pp. 18–22 (page 108).

[147] Tsungnan Lin, Chiapin Wang, and Po-Chiang Lin. "A neural-network-based context-aware handoff algorithm for multimedia computing." In: *ACM Transactions on Multimedia Computing, Communications, and Applications* 4.3. 2008, pp. 1–23 (page 115).

[148] Anders Lindgren, Avri Doria, and Olov Schelén. "Probabilistic routing in intermittently connected networks." In: *ACM SIGMOBILE mobile computing and communications review* 7.3. 2003, pp. 19–20 (page 11).

[149] Chen-Feng Liu, Mehdi Bennis, and H Vincent Poor. "Latency and reliability-aware task offloading and resource allocation for mobile edge computing." In: *IEEE Globecom Workshops*. IEEE. 2017, pp. 1–7 (pages 65, 66).

[150] Li Liu and Qi Fan. "Resource allocation optimization based on mixed integer linear programming in the multi-cloudlet environment." In: *IEEE Access* 6. 2018, pp. 24533–24542 (pages 72, 73).

[151] Qiang Liu, Siqi Huang, Johnson Opadere, and Tao Han. "An edge network orchestrator for mobile augmented reality." In: *IEEE Conference on Computer Communications*. IEEE. 2018, pp. 756–764 (page 87).

[152] Hong Lu, Jun Yang, Zhigang Liu, Nicholas D Lane, Tanzeem Choudhury, and Andrew T Campbell. "The jigsaw continuous sensing engine for mobile phone applications." In: *8th ACM Conference on Embedded Networked Sensor Systems*. 2010, pp. 71–84 (page 161).

[153] David C Luckham. "Event processing for business: Organizing the real-time enterprise". John Wiley & Sons, 2011 (page 20).

[154] Li Ma, Fei Yu, Victor CM Leung, and Tejinder Randhawa. "A new method to support UMTS/WLAN vertical handover using SCTP." In: *IEEE Wireless Communications* 11.4. 2004, pp. 44–51 (page 116).

[155] Samuel R Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. "TinyDB: An acquisitional query processing system for sensor networks." In: *ACM Transactions on database systems* 30.1. 2005, pp. 122–173 (page 161).

[156] Geoffrey Mainland, Greg Morrisett, and Matt Welsh. "Flask: Staged functional programming for sensor networks." In: *13th ACM SIGPLAN International Conference on Functional Programming*. 2008, pp. 335–346 (page 138).

[157] Evangelos Maltezos, Lazaros Karagiannidis, Thanasis Douklias, Aris Dadoukis, Angelos Amditis, and Evangelos Sdongos. "Preliminary design of a multipurpose UAV situational awareness platform based on novel computer vision and machine learning techniques." In: *5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference*. IEEE. 2020, pp. 1–8 (page 36).

[158] Asmae Ait Mansour, Nourddine Enneya, and Mohamed Ouadou. "A seamless handover based MIH-assisted PMIPV6 in heterogeneous network (LTE-WIFI)." In: *2nd international Conference on Big Data, Cloud and Applications*. 2017, pp. 1–5 (page 115).

[159] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. "A survey on mobile edge computing: The communication perspective." In: *IEEE Communications Surveys & Tutorials* 19.4. 2017, pp. 2322–2358 (page 47).

[160] Martina Marjanović, Lea Skorin-Kapov, Krešimir Pripužić, Aleksandar Antonić, and Ivana Podnar Žarko. "Energy-aware and quality-driven sensor management for green mobile crowd sensing." In: *Journal of Network and Computer Applications* 59. 2016, pp. 95–108 (page 161).

[161] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. "OpenFlow: Enabling innovation in campus networks." In: *ACM SIGCOMM Computer Communication Review* 38.2. 2008, pp. 69–74 (pages 137, 138).

[162] Alex McMahon and Stephen Farrell. "Delay-and disruption-tolerant networking." In: *IEEE Internet Computing* 13.6. 2009, pp. 82–87 (page 177).

[163] Amardeep Mehta, William Tärneberg, Cristian Klein, Johan Tordsson, Maria Kihl, and Erik Elmroth. "How beneficial are intermediate layer data centers in mobile edge networks?" In: *IEEE 1st International Workshops on Foundations and Applications of Self* Systems*. IEEE. 2016, pp. 222–229 (page 58).

[164] Manan Mehta. "ESP 8266: A breakthrough in wireless sensor networks and Internet of Things." In: *International Journal of Electronics and Communication Engineering & Technology* 6.8. 2015, pp. 7–11 (page 129).

[165] Ulrich Meissen, Stefan Pfennigschmidt, Agnes Voisard, and Tjark Wahnfried. "Context- and situation-awareness in information logistics." In: *International Conference on Extending Database Technology*. Springer. 2004, pp. 335–344 (page 34).

[166] 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: *International Conference on Mobile Computing, Applications, and Services*. Springer. 2015, pp. 12–29 (page 87).

[167] Guowang Miao, Jens Zander, Ki Won Sung, and Slimane Ben Slimane. "Fundamentals of mobile data networks". Cambridge University Press, 2016 (page 13).

[168] Emiliano Miluzzo, Nicholas D Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B Eisenman, Xiao Zheng, and Andrew T Campbell. "Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application." In: *6th ACM Conference on Embedded Network Sensor Systems*. 2008, pp. 337–350 (page 161).

[169] Dorian Minarolli, Artan Mazrekaj, and Bernd Freisleben. "Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing." In: *Journal of Cloud Computing* 6.1. 2017, pp. 1–18 (pages 85, 86).

[170] Sourav Mondal, Goutam Das, and Elaine Wong. "CCOMPASSION: A hybrid cloudlet placement framework over passive optical access networks." In: *IEEE Conference on Computer Communications*. IEEE. 2018, pp. 216–224 (pages 72, 73).

[171] Ehsan Moradi-Pari, Amin Tahmasbi-Sarvestani, and Yaser P Fallah. "A hybrid systems approach to modeling real-time situation-awareness component of networked crash avoidance systems." In: *IEEE Systems Journal* 10.1. 2014, pp. 169–178 (page 36).

[172] Darius Morawiec. "sklearn-porter." Transpile trained Scikit-learn estimators to C, Java, JavaScript and others (page 109).

[173] Luis Moreira-Matias, Joao Gama, Michel Ferreira, Joao Mendes-Moreira, and Luis Damas. "Predicting taxi–passenger demand using streaming data." In: *IEEE Transactions on Intelligent Transportation Systems* 14.3. 2013, pp. 1393–1402 (page 87).

[174] Abderrahmen Mtibaa, Afnan Fahim, Khaled A Harras, and Mostafa H Ammar. "Towards resource sharing in mobile device clouds: Power balancing across mobile devices." In: *ACM SIGCOMM Computer Communication Review*. Vol. 43. 4. ACM. 2013, pp. 51–56 (page 100).

[175] Anwesha Mukherjee, Debashis De, and Deepsubhra Guha Roy. "A power and latency aware cloudlet selection strategy for multi-cloudlet environment." In: *IEEE Transactions on Cloud Computing* 7.1. 2016, pp. 141–154 (pages 72, 73).

[176] Roger B Myerson and Mark A Satterthwaite. "Efficient mechanisms for bilateral trading." In: *Journal of Economic Theory* 29.2. 1983, pp. 265–281 (page 61).

[177] John F Nash Jr. "The bargaining problem." In: *Econometrica: Journal of the Econometric Society*. 1950, pp. 155–162 (page 54).

[178] Nidal Nasser, Sghaier Guizani, and Eyhab Al-Masri. "Middleware vertical handoff manager: A neural network-based solution." In: *IEEE International Conference on Communications*. IEEE. 2007, pp. 5671–5676 (page 115).

[179] Tim Nelson, Andrew D. Ferguson, Michael JG Scheer, and Shriram Krishnamurthi. "Tierless programming and reasoning for software-defined networks." In: *11th USENIX Symposium on Networked Systems Design and Implementation*. 2014, pp. 519–531 (page 138).

[180] Duong Tung Nguyen, Long Bao Le, and Vijay Bhargava. "Price-based resource allocation for edge computing: A market equilibrium approach." In: *IEEE Transactions on Cloud Computing*. 2018 (pages 72, 73).

[181] Thanh Nguyen, Chee Peng Lim, Ngoc Duy Nguyen, Lee Gordon-Brown, and Saeid Nahavandi. "A review of situation-awareness assessment approaches in aviation environments." In: *IEEE Systems Journal* 13.3. 2019, pp. 3590–3603 (page 36).

[182] Anthony J Nicholson and Brian D Noble. "Breadcrumbs: Forecasting mobile connectivity." In: *14th ACM International Conference on Mobile Computing and Networking*. 2008, pp. 46–57 (page 116).

[183] OpenFog Consortium. "OpenFog reference architecture for fog computing". Tech. rep. OpenFog Consortium, 2017 (page 26).

[184] Tao Ouyang, Rui Li, Xu Chen, Zhi Zhou, and Xin Tang. "Adaptive user-managed service placement for mobile edge computing: An online learning approach." In: *IEEE Conference on Computer Communications*. IEEE. 2019, pp. 1468–1476 (pages 72, 73).

[185] Christoph Paasch, Gregory Detal, Fabien Duchene, Costin Raiciu, and Olivier Bonaventure. "Exploring mobile/Wi-Fi handover with multipath TCP." In: *ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design*. 2012, pp. 31–36 (page 116).

[186] Susanna Paasovaara, Pradthana Jarusriboonchai, and Thomas Olsson. "Understanding collocated social interaction between Pokémon GO players." In: *16th International Conference on Mobile and Ubiquitous Multimedia*. 2017, pp. 151–163 (page 86).

[187] Mookyu Park, Haengrok Oh, and Kyungho Lee. "Security risk measurement for information leakage in IoT-based smart homes from a situational awareness perspective." In: *Sensors* 19.9. 2019, p. 2148 (page 36).

[188] Milan Patel, Brian Naughton, Caroline Chan, Nurit Sprecher, Sadayuki Abeta, Adrian Neal, et al. "Mobile-edge computing introductory technical white paper." In: *White Paper, Mobile-Edge Computing (MEC) Industry Initiative* 29. 2014, pp. 854–864 (page 28).

[189] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. "Scikit-learn: Machine learning in Python." In: *Journal of Machine Learning Research* 12.Oct. 2011, pp. 2825–2830 (page 108).

[190] Kai Peng, Xingda Qian, Bohai Zhao, Kejia Zhang, and Yang Liu. "A new cloudlet placement method based on affinity propagation for cyber-physical-social systems in wireless metropolitan area networks." In: *IEEE Access* 8. 2020, pp. 34313–34325 (pages 72, 73).

[191] Alvar Penning, Lars Baumgärtner, Jonas Höchst, Artur Sterz, Mira Mezini, and Bernd Freisleben. "DTN7: An open-Source disruption-tolerant networking implementation of Bundle Protocol 7." In: *International Conference on Ad-Hoc Networks and Wireless*. Springer. 2019, pp. 196–209 (pages xiii, 6, 11, 166).

[192] Charles Perkins et al. "IP mobility support for IPv4, revised." In: 2010 (page 116).

[193] Larry L Peterson and Bruce S Davie. "Computer networks: A systems approach". Elsevier, 2007 (pages 10, 23, 24).

[194] Marcus Pinnecke and Bastian Hoßbach. "Query optimization in heterogenous event processing federations." In: *Datenbank-Spektrum* 15.3. 2015, pp. 193–202 (page 161).

[195] Christopher Pluntke, Lars Eggert, and Niko Kiukkonen. "Saving mobile device energy with multipath TCP." In: *6th International Workshop on MobiArch*. 2011, pp. 1–6 (page 116).

[196] Jürgo S Preden, Kalle Tammemäe, Axel Jantsch, Mairo Leier, Andri Riid, and Emine Calis. "The benefits of self-awareness and attention in fog and mist computing." In: *Computer* 48.7. 2015, pp. 37–45 (page 34).

[197] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. "How hard can it be? designing and implementing a deployable multipath TCP." In: *9th USENIX symposium on networked systems design and implementation*. 2012, pp. 399–412 (page 116).

[198] Anuradha Ravi and Sateesh K Peddoju. "Mobile computation bursting: An application partitioning and offloading decision engine." In: *19th International Conference on Distributed Computing and Networking*. 46. ACM. 2018 (page 100).

[199] Andreas Reinhardt, Parag S Mogre, and Ralf Steinmetz. "Lightweight remote procedure calls for wireless sensor and actuator networks." In: *IEEE International Conference on Pervasive Computing and Communications Workshops*. IEEE. 2011, pp. 172–177 (page 190).

[200] Chenshan Ren, Xinchen Lyu, Wei Ni, Hui Tian, and Ren Ping Liu. "Profitable cooperative region for distributed online edge caching." In: *IEEE Transactions on Communications* 67.7. 2019, pp. 4696–4708 (pages 72, 73).

[201] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. "Fast context-aware recommendations with factorization machines." In: *34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2011, pp. 635–644 (page 86).

[202] Daniele Riboni and Claudio Bettini. "COSAR: Hybrid reasoning for context-aware activity recognition." In: *Personal and Ubiquitous Computing* 15.3. 2011, pp. 271–289 (page 86).

[203] Jonathan Rodriguez. "Fundamentals of 5G mobile networks". John Wiley & Sons, 2015 (page 14).

[204] Yoshitaka Sakurai and Takuo Watanabe. "Towards a statically scheduled parallel execution of an FRP language for embedded systems." In: *6th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems*. 2019, pp. 11–20 (page 137).

[205] Guido Salvaneschi, Sven Amann, Sebastian Proksch, and Mira Mezini. "An empirical study on program comprehension with reactive programming." In: *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. Hong Kong, China: ACM, 2014, pp. 564–575 (page 119).

[206] Guido Salvaneschi, Gerold Hintz, and Mira Mezini. "REScala: Bridging between object-oriented and functional style in reactive applications." In: *13th international Conference on Modularity*. 2014, pp. 25–36 (page 23).

[207] Guido Salvaneschi, Gerold Hintz, and Mira Mezini. "REScala: Bridging between object-oriented and functional style in reactive applications." In: *13th International Conference on Modularity*. 2014, pp. 25–36 (page 121).

[208] Guido Salvaneschi, Sebastian Proksch, Sven Amann, Sarah Nadi, and Mira Mezini. "On the positive effect of reactive programming on software comprehension: An empirical study." In: *IEEE Transactions on Software Engineering* 43.12. 2017, pp. 1125–1143 (page 119).

[209] Francisco Sant'Anna, Roberto Ierusalimschy, Noemi Rodriguez, Silvana Rossetto, and Adriano Branco. "The design and implementation of the synchronous language céu." In: *ACM Transactions on Embedded Computing Systems* 16.4. 2017, pp. 1–26 (page 138).

[210] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. "The case for VM-based cloudlets in mobile computing." In: *IEEE Pervasive Computing* 8.4. 2009, pp. 14–23 (pages 49, 87).

[211] Kensuke Sawada and Takuo Watanabe. "Emfrp: A functional reactive programming language for small-scale embedded systems." In: *C15th International Conference on Modularity*. 2016, pp. 36–44 (page 137).

[212] Lorenz Schauer, Martin Werner, and Philipp Marcus. "Estimating crowd densities and pedestrian flows using Wi-Fi and Bluetooth." In: *11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. 2014, pp. 171–177 (page 172).

[213] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. "Nexmon: Build your own Wi-Fi testbeds with low-level MAC and PHY access using firmware patches on off-the-shelf mobile devices." In: *11th Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*. Snowbird, UT, USA: ACM, 2017, pp. 59–66 (page 129).

[214] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. "Nexmon: The C-based firmware patching framework". 2017. URL: `https://nexmon.org` (visited on 02/01/2020) (page 129).

[215] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. "Using NexMon, the C-based Wi-Fi firmware modification framework." In: *9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 2016, pp. 213–215 (page 129).

[216] Julius Schulz-Zander, Carlos Mayer, Bogdan Ciobotaru, Raphael Lisicki, Stefan Schmid, and Anja Feldmann. "Unified programmability of virtualized network functions and software-defined wireless networks." In: *IEEE Transactions on Network and Service Management* 14.4. 2017, pp. 1046–1060 (page 137).

[217] Julius Schulz-Zander, Carlos Mayer, Bogdan Ciobotaru, Stefan Schmid, and Anja Feldmann. "OpenSDWN: Programmatic control over home and enterprise Wi-Fi." In: *1st ACM SIGCOMM Symposium on Software-defined Networking Research*. Santa Clara, CA, USA: ACM, 2015, 16:1–16:12 (page 137).

[218] Julius Schulz-Zander, Carlos Mayer, Bogdan Ciobotaru, Stefan Schmid, Anja Feldmann, and Roberto Riggio. "Programming the home and enterprise Wi-Fi with OpenSDWN." In: *ACM SIGCOMM Computer Communication Review*. Vol. 45. London, United Kingdom: ACM, 2015, pp. 117–118 (page 137).

[219] Keith Scott and Scott Burleigh. "Bundle protocol specification." In: 2007 (page 11).

[220] Russell Shackelford, Andrew McGettrick, Robert Sloan, Heikki Topi, Gordon Davies, Reza Kamali, James Cross, John Impagliazzo, Richard LeBlanc, and Barry Lunt. "Computing curricula 2005: The overview report." In: *ACM SIGCSE Bulletin* 38.1. 2006, pp. 456–457 (page 25).

[221] Huniya Shahid, Munam Ali Shah, Ahmad Almogren, Hasan Ali Khattak, Ikram Ud Din, Neeraj Kumar, and Carsten Maple. "Machine learning-based mist computing enabled Internet of battlefield things." In: *ACM Transactions on Internet Technology* 21.4. 2021, pp. 1–26 (page 119).

[222] Haichen Shen, Aruna Balasubramanian, Anthony LaMarca, and David Wetherall. "Enhancing mobile apps to use sensor hubs without programmer effort." In: *ACM International Joint Conference on Pervasive and Ubiquitous Computing*. 2015, pp. 227–238 (page 161).

[223] Cong Shi, Vasileios Lakafosis, Mostafa H Ammar, and Ellen W Zegura. "Serendipity: Enabling remote computing among intermittently connected mobile devices." In: *13th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM. 2012, pp. 145–154 (pages 100, 190).

[224] 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 (page 47).

[225] Yuan-Yao Shih, Chih-Yu Wang, and Ai-Chun Pang. "Fog computing service provision using bargaining solutions." In: *IEEE Transactions on Services Computing*. 2019 (pages 54, 72).

[226] Choonsung Shin, Jin-Hyuk Hong, and Anind K Dey. "Understanding and prediction of mobile application usage for smart phones." In: *ACM conference on ubiquitous computing*. 2012, pp. 173–182 (page 86).

[227] Ali Al-Shuwaili and Osvaldo Simeone. "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications." In: *IEEE Wireless Communications Letters* 6.3. 2017, pp. 398–401 (pages 65, 66).

[228] Nagaraj Shyam, Craig Harmer, and Ken Beck. "Managing remote procedure calls when a server is unavailable". US Patent 9,141,449. 2015 (page 190).

[229] Marin Silic, Goran Delac, Ivo Krka, and Sinisa Srbljic. "Scalable and accurate prediction of availability of atomic web services." In: *IEEE Transactions on Services Computing* 7.2. 2013, pp. 252–264 (page 85).

[230] Brian Sipos, Michael Demmer, Joerg Ott, and Simon Perreault. "Delay-tolerant networking TCP convergence layer protocol version 4". Tech. rep. IETF, 2019 (page 11).

[231] Markus Sommer, Jonas Höchst, Artur Sterz, Alvar Penning, and Bernd Freisleben. "ProgDTN: Programmable disruption-tolerant networking." In: *10th International Conference on Networked Systems*. Springer, 2022, to appear (page 6).

[232] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. "Spray and wait: An efficient routing scheme for intermittently connected mobile networks." In: *ACM SIGCOMM workshop on Delay-Tolerant Networking*. 2005, pp. 252–259 (page 11).

[233] Artur Sterz, Lars Baumgärtner, Jonas Höchst, Patrick Lampe, and Bernd Freisleben. "OPPLOAD: Offloading computational workflows in opportunistic networks." In: *IEEE 44th Conference on Local Computer Networks*. IEEE. 2019, pp. 381–388 (pages xiii, xiv, 6, 89, 91, 165, 166).

[234] Artur Sterz, Lars Baumgärtner, Ragnar Mogk, Mira Mezini, and Bernd Freisleben. "DTN-RPC: Remote procedure calls for disruption-tolerant networking." In: *IFIP Networking Conference and Workshops*. IEEE. 2017, pp. 1–9 (pages xiv, 6, 101, 165, 178).

[235] Artur Sterz, Matthias Eichholz, Ragnar Mogk, Lars Baumgärtner, Pablo Graubner, Matthias Hollick, Mira Mezini, and Bernd Freisleben. "ReactiFi: Reactive programming of Wi-Fi firmware on mobile devices." In: *The Art, Science, and Engineering of Programming* 5.2. 2020, pp. 4–1 (pages xiii, xiv, 119, 121, 165, 166).

[236] Artur Sterz, Patrick Felka, Bernd Simon, Sabrina Klos, Anja Klein, Oliver Hinz, and Bernd Freisleben. "Multi-stakeholder service placement via iterative bargaining with incomplete information." In: *IEEE/ACM Transactions on Networking*. 2022 (pages xiii, 47, 49).

[237] Randall Stewart and Christopher Metz. "SCTP: New transport protocol for TCP/IP." In: *IEEE Internet Computing* 5.6. 2001, pp. 64–69 (page 10).

[238] Denny Stohr, Alexander Frömmgen, Jan Fornoff, Michael Zink, Alejandro Buchmann, and Wolfgang Effelsberg. "QoE analysis of DASH cross-layer dependencies by extensive network emulation." In: *Workshop on QoE-based Analysis and Management of Data Communication Networks*. 2016, pp. 25–30 (page 112).

[239] Patrick Stuedi, Animesh Trivedi, Bernard Metzler, and Jonas Pfefferle. "Darpc: Data center RPC." In: *ACM Symposium on Cloud Computing*. 2014, pp. 1–13 (page 190).

[240] Kohei Suzuki, Kanato Nagayama, Kensuke Sawada, and Takuo Watanabe. "CFRP: A functional reactive programming language for small-scale embedded systems." In: *Theory and Practice of Computation*. 2017, pp. 1–13 (page 137).

[241] Telefonaktiebolaget L. M. Ericsson (Ericsson). "Ericsson mobility report". Report. Telefonaktiebolaget L. M. Ericsson (Ericsson), 2021 (page 1).

[242] Paul Thagard. "Cognitive science". Routledge, 2008 (page 36).

[243] Ilenia Tinnirello, Giuseppe Bianchi, Pierluigi Gallo, Domenico Garlisi, Francesco Giuliano, and Francesco Gringoli. "Wireless MAC processors: Programming MAC protocols on commodity hardware." In: *IEEE INFOCOM*. Orlando, FL, USA: IEEE, 2012, pp. 1269–1277 (page 137).

[244] Viet-Hoang Tran and Olivier Bonaventure. "Beyond socket options: Making the Linux TCP stack truly extensible." In: *IFIP Networking Conference*. IEEE, 2019, pp. 1–9 (page 136).

[245] Sacha Trifunovic, Sylvia T Kouyoumdjieva, Bernhard Distl, Ljubica Pajevic, Gunnar Karlsson, and Bernhard Plattner. "A decade of research in opportunistic networks: Challenges, relevance, and future directions." In: *IEEE Communications Magazine* 55.1. 2017, pp. 168–173 (page 12).

[246] Jianwei Tu and Christopher Stewart. "Replication for predictability in a Java RPC framework." In: *IEEE International Conference on Autonomic Computing*. IEEE. 2015, pp. 163–164 (page 189).

[247] Fadi Al-Turjman, Enver Ever, and Hadi Zahmatkesh. "Small cells in the forthcoming 5G/IoT: Traffic modelling and deployment overview." In: *IEEE Communications Surveys & Tutorials* 21.1. 2018, pp. 28–65 (page 167).

[248] Amin Vahdat, David Becker, et al. "Epidemic routing for partially connected ad hoc networks". 2000 (page 11).

[249] Andreas Voellmy and Paul Hudak. "Nettle: Taking the sting out of programming network routers." In: *International Symposium on Practical Aspects of Declarative Languages*. Springer. 2011, pp. 235–249 (page 138).

[250] Andreas Voellmy, Hyojoon Kim, and Nick Feamster. "Procera: A language for high-level reactive network control." In: *1st Workshop on Hot Topics in Software-defined Networks*. 2012, pp. 43–48 (page 138).

[251] Weetit Wanalertlak, Ben Lee, Chansu Yu, Myungchul Kim, Seung-Min Park, and Won-Tae Kim. "Behavior-based mobility prediction for seamless handoffs in mobile wireless networks." In: *Wireless Networks* 17.3. 2011, pp. 645–658 (page 116).

[252] Chuanmeizhi Wang, Yong Li, and Depeng Jin. "Mobility-assisted opportunistic computation offloading." In: *IEEE Communications Letters* 18.10. 2014, pp. 1779–1782 (page 101).

[253] Nan Wang, Blesson Varghese, Michail Matthaiou, and Dimitrios S Nikolopoulos. "ENORM: A framework for edge node resource management." In: *IEEE Transactions on Services Computing* 13.6. 2020, pp. 1086–1099 (pages 47, 65, 66).

[254] Sheng Wang and Takuo Watanabe. "Functional reactive EDSL with asynchronous execution for resource-constrained embedded systems." In: *International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. Springer. 2019, pp. 171–190 (page 137).

[255] Segev Wasserkrug, Avigdor Gal, Opher Etzion, and Yulia Turchin. "Complex event processing over uncertain data." In: *2nd International Conference on Distributed Event-Based Systems*. 2008, pp. 253–264 (page 195).

[256] Takuo Watanabe. "A simple context-oriented programming extension to an FRP language for small-scale embedded systems." In: *10th International Workshop on Context-Oriented Programming: Advanced Modularity for Run-time Composition*. 2018, pp. 23–30 (page 137).

[257] Daniel Winograd-Cort and Paul Hudak. "Settable and non-interfering signal functions for FRP: How a first-order switch is more than enough." In: *ACM SIGPLAN Notices* 49.9. 2014, pp. 213–225 (page 130).

[258] Thomas Wirtgen, Cyril Dénos, Quentin De Coninck, Mathieu Jadin, and Olivier Bonaventure. "The case for pluginized routing protocols." In: *IEEE 27th International Conference on Network Protocols*. IEEE. 2019, pp. 1–12 (page 136).

[259] Raymond K Wong, Victor W Chu, and Tianyong Hao. "Online role mining for context-aware mobile service recommendation." In: *Personal and Ubiquitous Computing* 18.5. 2014, pp. 1029–1046 (page 86).

[260] Mathieu Xhonneux, Fabien Duchene, and Olivier Bonaventure. "Leveraging eBPF for programmable network functions with IPv6 segment routing." In: *14th International Conference on Emerging Networking Experiments and Technologies*. ACM. 2018, pp. 67–72 (page 136).

[261] Anhao Xiang and Jun Zheng. "A situation-aware scheme for efficient device authentication in smart grid-enabled home area networks." In: *Electronics* 9.6. 2020, p. 989 (page 36).

[262] Yang Yan, Jianwei Huang, Xiaofeng Zhong, Ming Zhao, and Jing Wang. "Sequential bargaining in cooperative spectrum sharing: Incomplete information with reputation effect." In: *IEEE Global Telecommunications Conference*. IEEE. 2011, pp. 1–5 (pages 59, 72).

[263] Lei Yang, Jiannong Cao, Guanqing Liang, and Xu Han. "Cost aware service placement and load dispatching in mobile cloud systems." In: *IEEE Transactions on Computers* 65.5. 2015, pp. 1440–1452 (page 87).

[264] Lei Yang, Jiannong Cao, Zhenyu Wang, and Weigang Wu. "Network aware mobile edge computation partitioning in multi-user environments." In: *IEEE Transactions on Services Computing*. 2018 (page 101).

[265] Lei Yang, Jiannong Cao, Yin Yuan, Tao Li, Andy Han, and Alvin Chan. "A framework for partitioning and execution of data stream applications in mobile cloud computing." In: *ACM SIGMETRICS Performance Evaluation Review* 40.4. 2013, pp. 23–32 (page 101).

[266] Song Yang, Fan Li, Meng Shen, Xu Chen, Xiaoming Fu, and Yu Wang. "Cloudlet placement and task allocation in mobile edge computing." In: *IEEE Internet of Things Journal* 6.3. 2019, pp. 5853–5863 (pages 72, 73).

[267] Zhi Yang, Ben Y Zhao, Yuanjian Xing, Song Ding, Feng Xiao, and Yafei Dai. "AmazingStore: Available, low-cost online storage service using cloudlets." In: *IEEE International Parallel & Distributed Processing Symposium*. Vol. 10. 2010, pp. 2–2 (page 52).

[268] Juan Ye, Simon Dobson, and Susan McKeever. "Situation identification techniques in pervasive computing: A review." In: *Pervasive and Mobile Computing* 8.1. 2012, pp. 36–66 (page 34).

[269] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. "Fog computing: Platform and applications." In: *3rd IEEE workshop on hot topics in web systems and technologies*. IEEE. 2015, pp. 73–78 (page 89).

[270] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. "T-drive: Enhancing driving directions with taxi drivers' intelligence." In: *IEEE Transactions on Knowledge and Data Engineering* 25.1. 2011, pp. 220–232 (page 87).

[271] Nicholas Jing Yuan, Yu Zheng, Xing Xie, Yingzi Wang, Kai Zheng, and Hui Xiong. "Discovering urban functional zones using latent activity trajectories." In: *IEEE Transactions on Knowledge and Data Engineering* 27.3. 2014, pp. 712–725 (page 87).

[272] Alessandro Zanni, Se-Young Yu, Paolo Bellavista, Rami Langar, and Stefano Secci. "Automated selection of offloadable tasks for mobile computation offloading in edge computing." In: *13th international conference on network and service management*. IEEE. 2017, pp. 1–5 (page 101).

[273] Daniel Zhang, Yue Ma, Yang Zhang, Suwen Lin, X Sharon Hu, and Dong Wang. "A real-time and non-cooperative task allocation framework for social sensing applications in edge computing systems." In: *IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE. 2018, pp. 316–326 (page 101).

[274] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, KC Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. "Named data networking." In: *ACM SIGCOMM Computer Communication Review* 44.3. 2014, pp. 66–73 (page 195).

[275] Yang Zhang, Dusit Niyato, and Ping Wang. "Offloading in mobile cloudlet systems with intermittent connectivity." In: *IEEE Transactions on Mobile Computing* 14.12. 2015, pp. 2516–2529 (pages 101, 190).

[276] Yuan Zhang, Lei Jiao, Jinyao Yan, and Xiaojun Lin. "Dynamic service placement for virtual reality group gaming on mobile edge cloudlets." In: *IEEE Journal on Selected Areas in Communications* 37.8. 2019, pp. 1881–1897 (pages 72, 73).

[277] Yuanyuan Zhang, Wei Sun, and Yasushi Inoguchi. "CPU load predictions on the computational grid." In: *IEICE Transactions on Information and Systems* 90.1. 2007, pp. 40–47 (page 85).

[278] Yuyu Zhao, Guang Cheng, Yu Duan, Zhouchao Gu, Yuyang Zhou, and Lu Tang. "Secure IoT edge: Threat situation-awareness based on network traffic." In: *Computer Networks* 201. 2021, p. 108525 (page 36).

[279] Yu Zheng. "Trajectory data mining: An overview." In: *ACM Transactions on Intelligent Systems and Technology* 6.3. 2015, pp. 1–41 (page 87).

[280] 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, pp. 1–55 (page 87).

[281] Hubert Zimmermann. "OSI reference model-the ISO model of architecture for open systems interconnection." In: *IEEE Transactions on Communications* 28.4. 1980, pp. 425–432 (pages 9, 10).

Curriculum Vitae

This page contains personal data and is therefore not part of the online publication.