

Philipps



Universität  
Marburg

# ENERGY-EFFICIENT TRANSITIONAL NEAR-\* COMPUTING

**DISSERTATION**

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

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

Diplom-Informatiker  
**PABLO KARL GRAUBNER**  
geboren in Heidelberg

Marburg, im September 2018



Vom Fachbereich Mathematik und Informatik der Philipps-Universität Marburg  
(Hochschulkennziffer 1180) als Dissertation am 24. September 2018 angenommen.

- 1. Gutachter:** Prof. Dr.-Ing. Bernd Freisleben, Philipps-Universität Marburg
- 2. Gutachter:** Prof. Dr.-Ing. Ralf Steinmetz, Technische Universität Darmstadt

Tag der Einreichung am 20. September 2018.

Tag der mündlichen Prüfung am 4. Dezember 2018.



---

## **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.

---

Datum

---

Unterschrift



# Abstract

Studies have shown that communication networks, devices accessing the Internet, and data centers account for 4.6% of the worldwide electricity consumption. Although data centers, core network equipment, and mobile devices are getting more energy-efficient, the amount of data that is being processed, transferred, and stored is vastly increasing. Recent computer paradigms, such as fog and edge computing, try to improve this situation by processing data near the user, the network, the devices, and the data itself. In this thesis, these trends are summarized under the new term near-\* or near-everything computing. Furthermore, a novel paradigm designed to increase the energy efficiency of near-\* computing is proposed: transitional computing. It transfers multi-mechanism transitions, a recently developed paradigm for a highly adaptable future Internet, from the field of communication systems to computing systems. Moreover, three types of novel transitions are introduced to achieve gains in energy efficiency in near-\* environments, spanning from private Infrastructure-as-a-Service (IaaS) clouds, Software-defined Wireless Networks (SDWNs) at the edge of the network, Disruption-Tolerant Information-Centric Networks (DTN-ICNs) involving mobile devices, sensors, edge devices as well as programmable components on a mobile System-on-a-Chip (SoC). Finally, the novel idea of transitional near-\* computing for emergency response applications is presented to assist rescuers and affected persons during an emergency event or a disaster, although connections to cloud services and social networks might be disturbed by network outages, and network bandwidth and battery power of mobile devices might be limited.



# Deutsche Zusammenfassung

Studien haben gezeigt, dass Kommunikationsnetze, mobile Geräte und Rechenzentren bis zu 4,6% des weltweiten Stromverbrauchs ausmachen. Obwohl diese Geräte und Komponenten immer effizienter werden, steigt auch die Menge der Daten, die sie verarbeiten, übertragen und speichern. Neuere Paradigmen, wie z.B. Fog- und Edge Computing, erhöhen unter diesen Bedingungen die Effizienz von angebotenen Diensten, indem sie die Daten in der Nähe des Benutzers, des Netzwerks, der Endgeräte und an den Datenquellen selbst verarbeiten. In dieser Dissertation wird diese Entwicklung unter dem neuen Begriff Near-\* oder Near-Everything Computing zusammengefasst. Es wird darüber hinaus ein neuartiges Paradigma vorgeschlagen, das die Energieeffizienz von Near-\* Computing erhöhen soll: Transitional Computing. Es überträgt Multi-Mechanismen Transitionen, ein kürzlich entwickeltes Paradigma für ein hochgradig anpassungsfähiges zukünftiges Internet, aus dem Bereich der Kommunikationssysteme zu Near-\* Computing. In diesem Zusammenhang werden in dieser Dissertation drei neue Arten von Transitionen vorgestellt, sowie Ergebnisse in Bezug auf eine verbesserte Energieeffizienz präsentiert, unter anderem für Infrastructure-as-a-Service (IaaS) Clouds, für Software-definierte drahtlose Netzwerke (SDWNs) am Rande des Internets, in Information-Centric Networks (DTN-ICNs) mobiler Geräte, sowie auf programmierbaren Komponenten wie mobilen Co-Prozessoren. Ferner wird die neuartige Idee von Transitional Near-\* Computing für Anwendungen vorgestellt, die Rettungskräfte und betroffene Personen während eines Krisen- oder Katastrophenfalls unterstützen, obwohl Verbindungen zu Cloud-Diensten und sozialen Netzwerken durch Netzausfälle gestört sein könnten und die Netzwerkbandbreite sowie die Batterieleistung mobiler Geräte stark begrenzt sind.



# Acknowledgements

First of all, I would like to thank Prof. Dr. Bernd Freisleben for supervising me over the course of my dissertation, for his encouragement and assistance to advance this thesis.

I would also like to thank Prof. Dr. Ralf Steinmetz at the Technische Universität Darmstadt for kindly taking the time to review my thesis and for the opportunity to jointly work with researchers in the DFG SFB 1053.

I would like to thank my colleagues and students in the Distributed Systems Group in Marburg who were important for teaching and research projects I was involved in (in alphabetical order): Dr. Tim Dörnemann, Kay Dörnemann, Dr. Niels Fallenbeck, Patrick Heckmann, Jonas Höchst, Dr. Ernst Juhnke, Nik Korfhage, Patrick Lampe, Matthias Leinweber, Afef Mdhaffar, Dr. Markus Mühling, Marcel Müller, Nils Schmidt, Dominik Seiler, Markus Sommer, Artur Sterz and Christoph Thelen.

Furthermore, in the DFG SFB 1053 MAKI in Darmstadt, I benefited from the collaboration with Prof. Dr. Oliver Hinz, Prof. Dr. Matthias Hollick, Prof. Dr. Anja Klein, Prof. Dr. Mira Mezini, Prof. Dr. Guido Salvaneschi, Dr. Matthias Schultz, Patrick Felka, Sabrina Klos, Robin Klose, Mahdi Mousavi, and Daniel Wegemer. I would like to thank my colleague Michael Körber for the discussions about complex event processing, and Prof. Dr. Bernhard Seeger from the Database Research Group for his support.

Of particular importance for me and my decision to undertake a dissertation in the Distributed Systems Group was the support of Dr. Matthias Schmidt, who introduced me into the fascinating world of Unix. Furthermore, I really enjoyed the time with my long-term office colleague Dr. Roland Schwarzkopf, who is someone you can always ask for help if you have a Unix problem, and with Lars Baumgärtner, who did a lot of interesting work with Serval and emergency networks. Thank you, guys, it was nice to work with you. I also would like to thank Mechthild Kessler for taking care of almost everything. She has been the good soul of the working group.

I would like to thank my proofreaders Richard Williams, Dr. Matthias Schmidt, and Dr. Roland Schwarzkopf for reading parts of my thesis.

I also would like to thank my parents for supporting my studies, and my parents-in-law for helping us out with our sons Jannik and Leon when they were younger. Thea, Jannik and Leon, you are so important to me. I would like to thank you, Thea, for your love, support, and patience.



# My Contributions

As already indicated in my acknowledgments, I am very grateful that I had the opportunity to work with several persons who influenced my research in one way or another. Research in distributed computing paradigms and computer networks is, after all, a joint effort between different players. Publications evolve collaboratively, implementations are often planned and developed jointly, and intellectual forces are joined to work out novel concepts, formalize ideas, and discuss the results. Furthermore, students play a vital role in implementing ideas or assisting with experimental evaluations. Therefore, pinning achievements to a single contributor is not always possible. Since this thesis contains content from original publications, often in verbatim form, it also includes joint and sometimes practically indivisible contributions from colleagues. Therefore, I try to highlight my specific contributions as good as possible below.

Chapter 3 presents solely my views and ideas for the research topics addressed in this thesis, the concept of transitional near-\* computing is my original contribution.

Section 4.2 is based on the publications [GSF11; GSF13] that emerged from my work. The design presented there was intensively discussed with Matthias Schmidt and Bernd Freisleben. Both of them also reviewed, commented, and edited the publications before submission. Section 4.3 is based on the publication [Gra+17]. The presented concept is genuinely my work. Markus Sommer was involved in the experimental evaluations and the implementation during his bachelor thesis. Matthias Hollick and Bernd Freisleben reviewed, commented, and edited the publication before submission.

Section 5.2 is based on the publication [GHF15]. The concept is genuinely my work, Patrick Heckmann contributed parts of the software implementation and conducted several of the experiments during his bachelor thesis, and Bernd Freisleben made suggestions and helped to write the paper. Section 5.3 is based on the publication [Gra+18a]. The concept and its integration into Serval is genuinely my work, Lars Baumgärtner contributed Serval-specific parts in Section 5.3.3, Patrick Lampe conducted experiments presented in Section 5.3.4.1, Jonas Höchst conducted and described the experiments presented in Section 5.3.4.2. The publication was intensively discussed with and reviewed by Mira Mezini and Bernd Freisleben before submission.

Section 6.2 is based on the publication [Gra+18b]. The concept is genuinely my work. Christoph Thelen implemented parts of the CQL frontend and prototyped the eBPF mode, Michael Körber contributed the mode selection algorithm (Section 6.2.2.4). Artur Sterz implemented the use case for gathering and analyzing mobility data in Section 6.2.4.4 and conducted most of the experiments. Bernhard Seeger, Guido Salvaneschi, Mira Mezini, and Bernd Freisleben reviewed, commented, and edited the publication before submission.

Section 6.3 is based on the publication [Ste+]. The concept was developed collaboratively by Artur Sterz and myself, but the design and implementation of the programming language frontend and the corresponding runtime presented in Sections 6.3.2 and 6.3.3 is my work. Artur Sterz, Matthias Schulz, Robin Klose, Daniel Wegemer and myself collaborated developing the use cases and resolving implementation issues. Matthias Schulz developed position independent code modules for firmware patching framework Nexmon. Artur Sterz implemented the use cases and conducted the experiments. Mira Mezini, Matthias Hollick, and Bernd Freisleben reviewed, commented, and edited the paper before submission.

---

Chapter 7 combines contributions that were previously published in: [Gra+17; Gra+18b; Gra+18a; Ste+]. The ideas and concepts are genuinely my work. The indoor localization approach presented in Section 7.6.2 was implemented by Artur Sterz. Markus Sommer was involved in performing the experiments in Section 7.4, Artur Sterz conducted experiments in Section 7.6.

# Contents

<b>Abstract</b>	<b>vii</b>
<b>Deutsche Zusammenfassung</b>	<b>ix</b>
<b>Acknowledgements</b>	<b>xi</b>
<b>My Contributions</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problem Statement . . . . .	2
1.3 Contributions of this Thesis . . . . .	3
1.4 Organization of this Thesis . . . . .	6
<b>2 Fundamentals</b>	<b>7</b>
2.1 Communication Networks . . . . .	7
2.1.1 OSI Reference Model . . . . .	7
2.1.2 Software-defined Networks . . . . .	9
2.1.3 Software-defined Wireless Networks . . . . .	10
2.2 Virtualization . . . . .	12
2.2.1 Virtual Machines . . . . .	12
2.2.2 Network Function Virtualization . . . . .	14
2.3 Computing Paradigms . . . . .	15
2.3.1 Cloud Computing . . . . .	15
2.3.2 Fog Computing . . . . .	16
2.3.3 Mobile Cloud Computing . . . . .	18
2.3.4 Mobile Edge Computing . . . . .	18
2.3.5 Comparison of Computing Paradigms . . . . .	19
2.4 Energy Efficiency . . . . .	21
2.4.1 Definitions . . . . .	21
2.4.2 CPU Power Management . . . . .	22
2.4.3 Data Center Power Management . . . . .	23
2.4.4 Mobile Phone Power Management . . . . .	24
2.5 Summary . . . . .	25
<b>3 Transitional Near-* Computing</b>	<b>27</b>
3.1 Near-* Computing . . . . .	27
3.1.1 Near-User . . . . .	27
3.1.2 Near-Network . . . . .	28

3.1.3	Near-Device . . . . .	28
3.1.4	Near-Data . . . . .	28
3.1.5	Definition of Near-* Computing . . . . .	28
3.2	Multi-Mechanism Adaptation . . . . .	29
3.2.1	Mechanism Transition . . . . .	29
3.2.2	Multi-Mechanisms . . . . .	30
3.3	Transitional Computing . . . . .	32
3.4	Energy-efficient Transitional Near-* Computing . . . . .	35
3.5	Summary . . . . .	37
<b>4</b>	<b>Transitions between Locations in Near-* Computing</b>	<b>39</b>
4.1	Motivation . . . . .	39
4.2	Virtual Machine Consolidation in IaaS-Clouds . . . . .	41
4.2.1	Computational Transitions . . . . .	41
4.2.2	Design . . . . .	42
4.2.3	Implementation . . . . .	47
4.2.4	Experimental Evaluation . . . . .	49
4.2.5	Related Work . . . . .	56
4.2.6	Summary . . . . .	57
4.3	Dynamic Role Assignment in SDWNs . . . . .	58
4.3.1	Computational Transitions . . . . .	58
4.3.2	Design . . . . .	59
4.3.3	Implementation . . . . .	61
4.3.4	Experimental Evaluation . . . . .	62
4.3.5	Related Work . . . . .	68
4.3.6	Summary . . . . .	70
4.4	Summary . . . . .	71
<b>5</b>	<b>Transitions between Implementations in Near-* Computing</b>	<b>73</b>
5.1	Motivation . . . . .	73
5.2	Data Sparsing for MapReduce . . . . .	75
5.2.1	Computational Transitions . . . . .	75
5.2.2	Design . . . . .	76
5.2.3	Implementation . . . . .	79
5.2.4	Experimental Evaluation . . . . .	82
5.2.5	Related Work . . . . .	87
5.2.6	Summary . . . . .	89
5.3	Opportunistic Named Functions . . . . .	90
5.3.1	Computational Transitions . . . . .	90
5.3.2	Design . . . . .	91
5.3.3	Implementation . . . . .	95
5.3.4	Experimental Evaluation . . . . .	98
5.3.5	Related Work . . . . .	102
5.3.6	Summary . . . . .	104
5.4	Summary . . . . .	105
<b>6</b>	<b>Transitions between Modes in Near-* Computing</b>	<b>107</b>
6.1	Motivation . . . . .	107

6.2	Multimodal Complex Event Processing . . . . .	109
6.2.1	Computational Transitions . . . . .	109
6.2.2	Design . . . . .	110
6.2.3	Implementation . . . . .	116
6.2.4	Experimental Evaluation . . . . .	121
6.2.5	Related Work . . . . .	128
6.2.6	Summary . . . . .	129
6.3	Reactive Programming of Wi-Fi Firmware on Mobile Devices . . . . .	131
6.3.1	Computational Transitions . . . . .	131
6.3.2	Design . . . . .	132
6.3.3	Implementation . . . . .	134
6.3.4	Experimental Evaluation . . . . .	138
6.3.5	Related Work . . . . .	149
6.3.6	Summary . . . . .	150
6.4	Summary . . . . .	151
<b>7</b>	<b>Transitional Near-* Computing for Emergency Response Applications</b>	<b>153</b>
7.1	Motivation . . . . .	153
7.2	Design . . . . .	154
7.3	Computational Transitions . . . . .	156
7.4	Dynamically Placed Resource-aware Gateways . . . . .	158
7.5	Search for Missing Persons . . . . .	159
7.6	Battery-efficient Context Sensing in Mobile Devices . . . . .	163
7.6.1	Physical Activity Detection . . . . .	163
7.6.2	Indoor Localization of Mobile Devices . . . . .	164
7.7	Summary . . . . .	168
<b>8</b>	<b>Conclusions</b>	<b>169</b>
8.1	Summary . . . . .	169
8.2	Future Work . . . . .	171
	<b>List of Figures</b>	<b>175</b>
	<b>List of Tables</b>	<b>179</b>
	<b>Bibliography</b>	<b>181</b>
	<b>Curriculum Vitae</b>	<b>199</b>



*“Integrated circuits will lead to such wonders as home computers—or at least terminals connected to a central computer—automatic controls for automobiles, and personal portable communications equipment.”*

G. E. Moore: Cramming More Components onto Integrated Circuits (1965)

# 1

## Introduction

About fifty years ago, Gordon E. Moore predicted technologies people are dealing with today in his groundbreaking article *Cramming More Components onto Integrated Circuits* [Moo65]. Due to low costs and simplified designs, he stated that integrated circuits will potentially enable manifold new devices in the future: home computers (connected to central computers), automatic controls for automobiles, and personal portable communications.

In fact, Moore vaguely anticipated the development of computing and communication equipment in the following decades. The Internet was developed during the 1970s, spread commercially in the 1990s, and was the technological basis for the rise of mobile devices after the millennium. Today, the number of Internet-connected devices is larger than the world’s population. By 2020, about 50 billion of Internet-connected devices are expected, which is more than six devices per person worldwide [Eva11].

The industry, however, has been and still is confronted with fundamental limits in terms of miniaturization of transistors, operating temperatures, and power consumption. The term “Moore’s Law” is more and more referenced within the phrase “the end of Moore’s Law” [TW17]. Although Moore’s prediction of the development of the semiconductor industry was correct, new paradigms in the design of hardware and software are needed today. The future Internet will rely on hardware specialization, on accelerators for I/O-devices, and on programmable network cards or solid state drives (SSDs) to achieve high-performance while reducing energy consumption. Future computing paradigms will need to find a solution that addresses integration and combination of these specialized architectures to perform computations efficiently near the data sources and near the users.

### 1.1 Motivation

In recent years, the industry has taken a lot of effort to make the Information and Communication Technology (ICT) sector greener. In particular, the largest Internet companies made investments to reduce their carbon footprint and to improve their data centers’ Power Usage Effectiveness (PUE). For example, between 2008 and 2018, Google improved the efficiency of their data center

computing equipment by 10%<sup>1</sup>, and other Internet companies have presented equal efficiency gains.

Nevertheless, studies have shown that communication networks, Internet access devices and data centers account for 4.6% of the worldwide electricity consumption [LP17]. Although data centers, core network equipment, and mobile devices are inarguably getting more efficient, the amount of data that is being processed, transferred, and stored is also vastly increasing. Predictions are hard to make on how the ICT share of the global electricity consumption is going to develop, since this is an ongoing trend. But studies on historical data have shown that the compound annual growth rate of key Internet equipment between 2007 and 2012 was about 7%, while the rate for general electricity consumption was about 3% [Hed+14]. In other words, in this period, the drivers for increased power consumption outweighed the drivers for higher energy efficiency.

This trend is not limited to the core network components, it also applies to battery-powered devices. Battery technology is determined and limited by its chemical properties that have not developed as fast as the power requirements of modern mobile devices in recent years [Tar+14]. For computing paradigms like fog and edge computing that leverage the computing power of battery-powered smartphones and even autonomous Internet-of-Things (IoT) sensors, battery lifetime is one of the most important problems. Within these edge-oriented paradigms, there is a need to orchestrate different edge device capabilities, to coordinate data processing on different layers, and to investigate new ways of power management and reduction [Li+18].

Taking this into account, the research problem of reducing the power consumption and increasing the efficiency of the Internet will not disappear. On the contrary, it will remain a relevant research field for a long time, since future computing paradigms will require novel approaches that are able to incorporate emerging hardware technologies, to allow dynamic application- and usage-driven adaptations in order to increase their efficiency, and to reduce their power consumption.

Developing such novel approaches is the high-level objective of this thesis. In the following, several associated problems and ways to solve them are presented.

## 1.2 Problem Statement

The aim of this thesis is to increase the energy efficiency in near-\* (near-everything) computing, where computations are performed close to the users, to the end devices, to the data, and to the network. In these environments, energy efficiency gains can be achieved by putting more hardware components into power save mode, by reducing their power consumption during execution, by adapting the resource usage during a computation, and by performing computations closer to the data and to the network, where they can be performed more energy-efficiently. These measures require adaptations of services or applications at the software level. Several aspects and subproblems of this approach are discussed below.

The first research question is how to configure, place and relocate computing instances in near-\* environments such that the power consumption of the involved components is reduced while the quality of service is maintained. Typically, computing instances are isolated environments

---

<sup>1</sup><https://www.google.com/about/datacenters/efficiency/internal>

and it is unknown which kind of software runs inside them. Therefore, adaptations have to be performed at the level of (virtualized) resources such as storage, network, and processing units.

The second research question is how to perform near-user and near-network computations energy-efficiently at the granularity of functions, modules, or classes. Programming and execution environments in near-\* computing typically provide a programming model, a programming interface (API), or a compiler for the application source code that can be leveraged for, e.g., adapting their computational complexity or the transmission of their input and output data via network.

The third research question is how to leverage specialized low-power components like sensor or Wi-Fi System-on-a-Chip co-processors on mobile and IoT devices for an increased energy efficiency. Instead of statically gluing together shards of microcontroller- and operating system code during compile-time, applications need to adapt to the capabilities of a device's accelerators and low-power processors at runtime. Therefore, applications need an interface to perform computations directly on these low-power components, skipping the kernel and/or the main CPU completely.

### 1.3 Contributions of this Thesis

In this thesis, the following contributions are presented.

- The novel idea of transitional computing is proposed. It makes use of mechanism transitions [Frö+16], a paradigm for a highly adaptive future Internet, and applies it to near-\* computing environments. In extension to the existing paradigm, novel types of transitions between computing mechanisms are identified: vertically on a host between the different levels of the software and hardware architecture, horizontally between different hosts, and between different realizations of an application.

These transitions are applied to different domains in order to realize the high-level objective of this thesis: energy-efficient computing.

First, transitions at the level of (virtualized) resources for infrastructures that provide computing instances like virtual servers or containers are presented.

- In an infrastructure-as-a-service cloud, energy savings are achieved by consolidating servers through virtual machine live migrations, considering the energy costs during storage synchronization and during live migration itself.
- In a wireless network, placements of network- and application services on wireless access points and on mobile devices are combined with topology transitions and transitions in the wireless network stack, reducing the overall energy consumption.

Second, transitions at the granularity of functions and modules of near-\* applications are presented.

- A novel approach for MapReduce, a programming model for distributed data processing in a cluster, allows transitions to be applied between different implementations of data stream operators in the distributed file system underneath MapReduce. Transitions are used to reduce processing times and to improve the energy efficiency of MapReduce applications that are robust to input variations while preserving the quality of the results.
- Opportunistic Named Functions (ONFs) perform computational transitions in a disruption-tolerant manner on mobile and edge devices. Users express interest in particular content and application-specific functions are then executed in the network on the fly, either partially or totally, in an opportunistic manner. Opportunistic named functions can reduce network congestion and improve battery lifetime in a network consisting of battery-powered sensors, mobile devices, and mobile routers, while delivering and processing information.

Third, transitions between functions running in different modes, i.e., on a CPU or on a specialized low-power processor, are presented.

- Multimodal Complex Event Processing (CEP) provides a high-level query language interface to filter, aggregate, and correlate event streams on a mobile device. It automatically breaks up CEP queries and performs computational transitions for the involved CEP operators in order to process streams of events on-device in different modes, providing significant improvements in terms of power consumption and throughput: 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).
- Reactive programming, an asynchronous programming paradigm for high-level languages with a focus on the propagation of changes, is presented for mobile and edge nodes. It is used in a software-defined network to perform transitions between the layers of execution of network services, either on the mobile CPU or on the Wi-Fi firmware, in order to reduce power consumption and improve latency considerably.

Finally, the concept of transitional computing is applied to a real-world scenario.

- Transitional computing is used to provide basic means of communication and computation and to support rescuers and affected people during an emergency event such as a flood or an earthquake. IT infrastructures in such scenarios need to operate on generator- or battery-powered devices with intermittent communication and scarce bandwidth. Computational transitions enable adaptive applications that help to prevent turning an emergency event into a disaster, although the Internet and cloud services are disrupted.

The following papers were published during the work on this thesis:

1. Artur Sterz, [Pablo Graubner](#), Matthias Schulz, Robin Klose, Daniel Wegemer, Matthias Hollick, Mira Mezini, Bernd Freisleben. *ReactiFi: Reactive Programming of Wi-Fi Firmware on Mobile Devices*. Submitted, 2018
2. [Pablo Graubner](#), Christoph Thelen, Michael Körber, Artur Sterz, Guido Salvaneschi, Mira Mezini, Bernhard Seeger, Bernd Freisleben. *Multimodal Complex Event Processing on Mobile Devices*. Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems (DEBS'18), pp. 112–123, ACM, 2018

3. Pablo Graubner, Patrick Lampe, Jonas Höchst, Lars Baumgärtner, Mira Mezini, Bernd Freisleben. *Opportunistic Named Functions in Disruption-tolerant Emergency Networks*. Proceedings of the ACM International Conference on Computing Frontiers International Conference on Computing Frontiers 2018 (CF'18), pp. 129–137, ACM, 2018
4. Pablo Graubner, Markus Sommer, Matthias Hollick, Bernd Freisleben. *Dynamic Role Assignment in Software-Defined Wireless Networks*. Proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC'17), pp. 760–766, IEEE Computer Society, 2017
5. Lars Baumgärtner, Pablo Graubner, Jonas Höchst, Anja Klein, Bernd Freisleben. *Speak Less, Hear Enough: On Dynamic Announcement Intervals in Wireless On-demand Networks*. Proceedings of the 13th Annual Conference on Wireless On-demand Network Systems and Services (WONS'17), pp. 33–40, IEEE Computer Society, 2017
6. Lars Baumgärtner, Paul Gardner-Stephen, Pablo Graubner, Jeremy Lakeman, Jonas Höchst, Patrick Lampe, Nils Schmidt, Stefan Schulz, Artur Sterz, Bernd Freisleben. *An Experimental Evaluation of Delay-tolerant Networking with Serval*. Proceedings of the IEEE Global Humanitarian Technology Conference, (GHTC'16), pp. 70–79, IEEE Computer Society, 2016
7. Pablo Graubner, Lars Baumgärtner, Patrick Heckmann, Marcel Müller, Bernd Freisleben. *Dynalyze: Dynamic Analysis of Mobile Apps in a Platform-as-a-Service Cloud*. Proceedings of the 8th IEEE Intl. Conference on Cloud Computing (CLOUD'15), pp. 925–932, IEEE Computer Society, 2015
8. Lars Baumgärtner, Pablo Graubner, Nils Schmidt, Bernd Freisleben. *AndroLyze: A Distributed Framework for Efficient Android App Analysis*. Proceedings of the 2015 IEEE International Conference on Mobile Services (MS'15), pp. 73–80, IEEE Computer Society, 2015
9. Pablo Graubner, Patrick Heckmann, Bernd Freisleben. *Energy-Efficient Data Processing Through Data Sparsing with Artifacts*. Proceedings of the 30th International Conference on High Performance Computing (ISC HPC'15), pp. 307–322, Springer, 2015
10. Pablo Graubner, Matthias Schmidt, Bernd Freisleben. *Energy-Efficient Virtual Machine Consolidation*. IT Professional, Volume 15, Number 2, pp. 28–34, IEEE Computer Society, 2013
11. Lars Baumgärtner, Pablo Graubner, Matthias Leinweber, Roland Schwarzkopf, Matthias Schmidt, Bernhard Seeger, Bernd Freisleben. *Mastering Security Anomalies in Virtualized Computing Environment via Complex Event Processing*. Proceedings of the The Fourth International Conference on Information, Process, and Knowledge Management (eKNOW'12), pp. 76–81, XPS, 2012
12. Pablo Graubner, Matthias Schmidt, Bernd Freisleben. *Energy-Efficient Management of Virtual Machines in Eucalyptus*. Proceedings of the 4th IEEE International Conference on Cloud Computing (CLOUD'11), pp. 243–250, IEEE Computer Society, 2011
13. Matthias Schmidt, Lars Baumgärtner, Pablo Graubner, David Böck, Bernd Freisleben. *Malware Detection and Kernel Rootkit Prevention in Cloud Computing Environments*. Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP'11), pp. 603–610, IEEE Computer Society, 2011

## 1.4 Organization of this Thesis

The thesis is organized as follows.

In Chapter 2, an overview of fundamental technologies used in this thesis is given and basic terms are defined.

The novel concept of transitional near-\* computing is introduced in Chapter 3. Transitional computing shifts the scope of mechanism transitions from communication systems to computing systems. The near-\* computing paradigm is introduced, transitional computing is defined, and an overview over the contributions to energy-efficient transitional near-\* computing is given.

In the following chapters, three types of computational transitions, each with two instances, are presented.

First, two types of transitions between locations are introduced in Chapter 4. A novel approach to virtual machine consolidation in Infrastructure-as-a-Service (IaaS) environments is introduced in Section 4.2. The novel concept of Dynamic Role Assignment in Software-defined Wireless Networks is presented in Section 4.3. It allows services to be combined with network topology transitions.

The following Chapter 5 contains two instances of transitions between implementations. The novel approach of multimodal MapReduce is presented in Section 5.2. It allows MapReduce clusters to apply transitions between different implementations of data stream operators in the Hadoop distributed file system (HDFS) underneath MapReduce. Opportunistic Named Functions (ONFs) are introduced in Section 5.3. Users express interest in particular content and application-specific functions are then executed in the network on the fly, either partially or totally, in an opportunistic manner.

Furthermore, two instances of transitions between functions running in different modes, i.e., on a CPU or on a specialized low-power processor are presented in Chapter 6. Multimodal Complex Event Processing (CEP) is introduced in Section 6.2. Application-specific queries on heterogeneous event streams are executed on mobile hardware/software architectures in an energy-efficient manner. Reactive Programming on Wi-Fi Firmwares (ReactiFi) is presented in Section 6.3. Reactive programs are developed in a high-level language and dynamically loaded to the Wi-Fi firmware of mobile devices.

In Chapter 7, transitional computing is applied to a real-world scenario. It is used to support rescuers in an emergency event like a flood or an earthquake, i.e., in a network environment with limited battery power and network bandwidth.

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

# 2

## Fundamentals

This chapter introduces fundamental terms, concepts, and technologies used in this thesis. Section 2.1 explains communication networks. Section 2.2 describes the concept of virtualization. Section 2.3 introduces the computing paradigms of cloud, fog, mobile cloud and mobile edge computing. Section 2.4 defines energy efficiency and describes power management techniques for data centers and mobile phones. Finally, Section 2.5 summarizes this chapter.

### 2.1 Communication Networks

Communication networks are an important part of the technical basis for distributed computing systems. In the following, Section 2.1.1 describes the conceptual OSI reference model. Afterwards, Software-Defined Networks (SDNs) and Software-Defined Wireless Networks (SDWNs) are discussed.

#### 2.1.1 OSI Reference Model

The OSI (Open Systems Interconnection) reference model is a 7-layer architecture of protocols defined by the International Organization for Standardization (ISO) in 1977, reflecting the experiences with network protocols at that time. Basically, similar functions formerly implemented in proprietary services and protocols were merged into layers and abstracted with common interfaces. This abstraction allowed protocols within a layer to be changed or redesigned without modifying the interfaces in adjacent layers.

Figure 2.1 shows the OSI reference model according to [Zim80]. The lowest layer is the *physical layer* that permits the usage of various physical media. The *data link layer* represents the data link between nodes and the control procedures to access the medium, i.e., to reduce, detect and correct errors after the reception of a frame. Circuitry and software required to implement physical layer functions are also abbreviated with *PHY*, and a common sublayer to data link layer for medium access control is abbreviated with *MAC*. At the *network layer*, packets are transmitted between networks. Intermediate nodes act as forwarders or routers. The network layer provides an interface for transmitting packets from a source to a receiver without knowing the route of the respective packets.

The data transportation from source to destination is controlled by protocols within the *transport layer*. The transport layer provides logical abstractions for application processes to transfer data end-to-end. For example, reliable stream- and connection-oriented data transmission including packet flow and network congestion control is provided by the Transmission Control

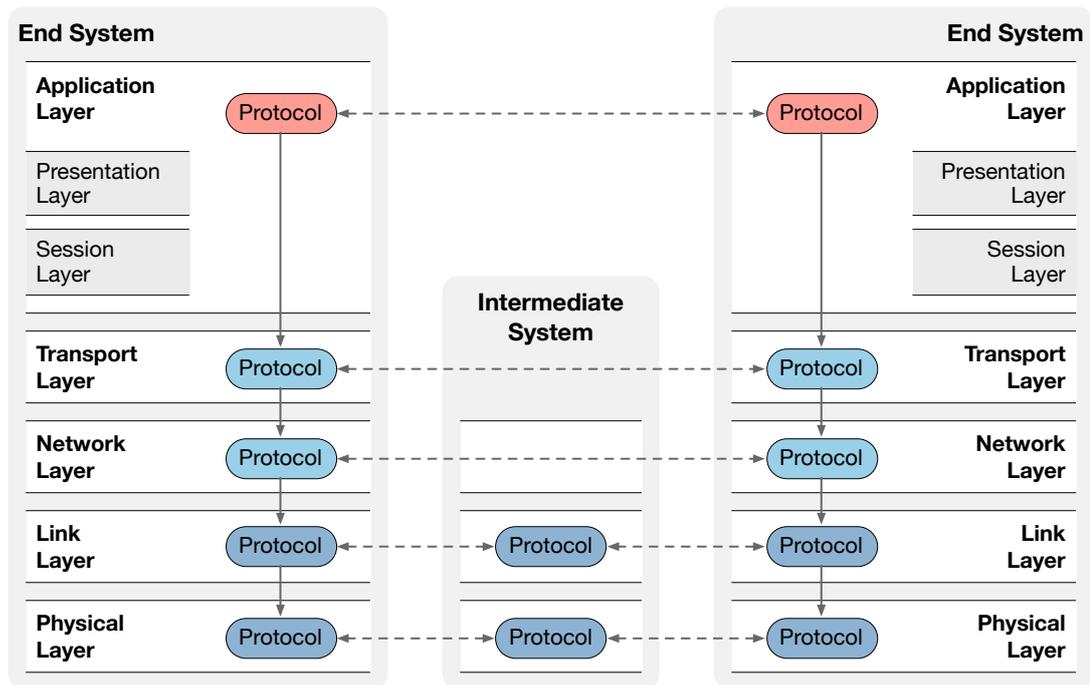


Figure 2.1 OSI reference model according to [Zim80].

Protocol (TCP) [Pos81], while datagram-oriented best-effort transmission is provided by the User Datagram Protocol (UDP) [Pos80]. Typically, layers residing from the transport layer downwards are implemented within the operating system of an end node.

The following two layers are often considered to be deprecated. The provisioning of connection management (*session*) is usually implemented within transport layer protocols. Furthermore, different data representations (*presentation*) are usually implemented within the topmost *application layer*.

The *application layer* is the closest layer to the end user and interacts directly with user applications. For example, the stateless application layer protocols Hypertext Transfer Protocol (HTTP), SPDY<sup>1</sup> and HTTP/2 [BPT15] realize the transfer of hypertext within the World Wide Web (WWW). Protocols in-between transport and application layer are provided as linkable libraries to applications and implements for example application-level authentication and encryption, such as the Transport Layer Security (TLS) [Res18] protocol. Furthermore, some custom protocols reimplement features from the transport layer for latency-critical or throughput-oriented applications. E.g., the QUIC (Quick UDP Internet Connections)<sup>2</sup> protocol supports multiplexed connections between endpoints and is used to improve the performance of Web browsers.

<sup>1</sup><http://www.chromium.org/spdy/spdy-whitepaper>

<sup>2</sup><http://www.chromium.org/quic>

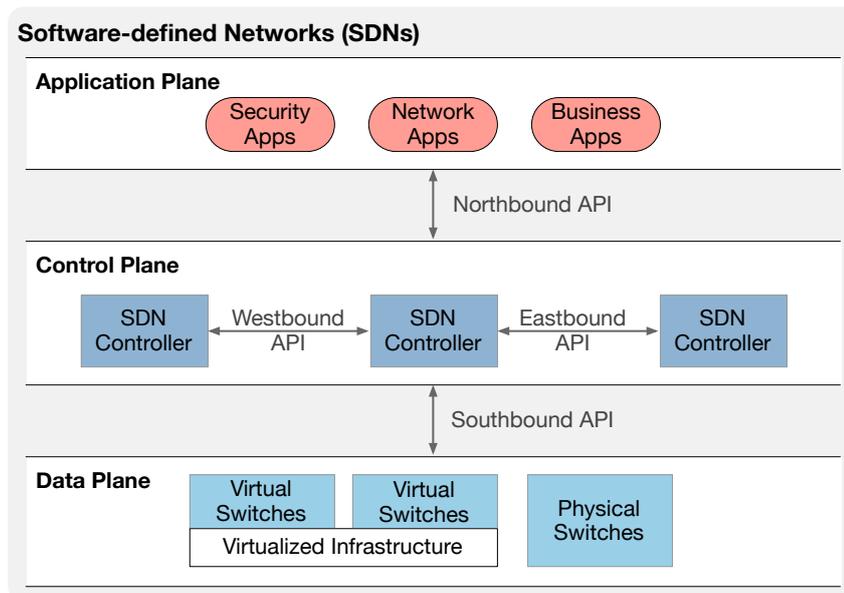


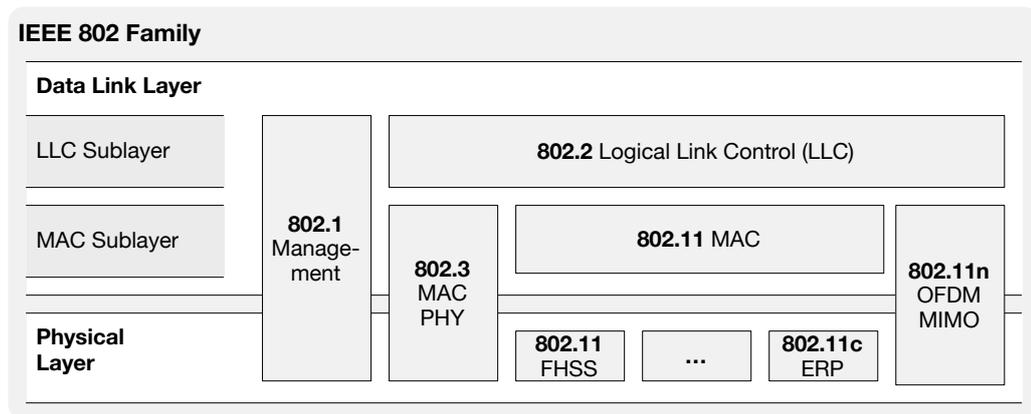
Figure 2.2 SDN architecture according to [Sta15]

### 2.1.2 Software-defined Networks

Traditional network architectures, like the aforementioned OSI reference model, are mostly static and complex. The Open Network Foundation (ONF) stated that within these complex architectures, policy changes are not automated and time consuming for network administrators and that they lack open interfaces for network functions, e.g., the automatic scaling of network resources to avoid over-subscription. In particular, traditional network architectures cannot compete with the following challenges: the *network demand* is increasing massively through cloud computing, big data processing, mobile traffic and Internet-of-Things (IoT). The *network supply* is rising through Ethernet/Wi-Fi operating at Gbps speeds, and 4G/5G for mobile devices. (iii) *Traffic patterns* are more complex since network traffic is not only exchanged vertically between a client and a server, but also horizontally between servers and between clients. Traffic is also less predictable due to modern applications (e.g., voice, video, data). The *imposed load* has a growing complexity, variability, and high volume [Fou12b].

These challenges are addressed in Software-defined Networks (SDNs) at the network architecture level by decoupling the control- from the data plane: The *data plane* summarizes hardware components and software responsible for packet forwarding on an intermediate node, while the *control plane* denotes the administration of the data plane and the determination of a routing path. In SDN, network control and state are centralized logically, and the underlying network infrastructure is accessed via an abstract interface [Fou12b].

Figure 2.2 shows the SDN architecture according to Stallings [Sta15]. At the bottom, the data plane consists of virtual and physical switches, both responsible for forwarding packets. The data plane is controlled via a public interface (southbound API), such as OpenFlow [Fou12a]. On the one hand, OpenFlow is a protocol between SDN controllers and network devices. On the other hand, it specifies the logical structure of the network switch functionality. In the middle of Figure 2.2, the control plane consists of a federation of controllers that instruct the underlying switches in the data plane, i.e., leveraging information about capacity and demand obtained



**Figure 2.3** The IEEE 802 family and its relation to the OSI model according to [Gas05].

from the data plane. At the top, *network applications* are communicating via the northbound API with the control plane. The key idea here is to abstract the interiors of the network, in such a way that network application developers can define changes to the network without having to understand the details of its realization.

### 2.1.3 Software-defined Wireless Networks

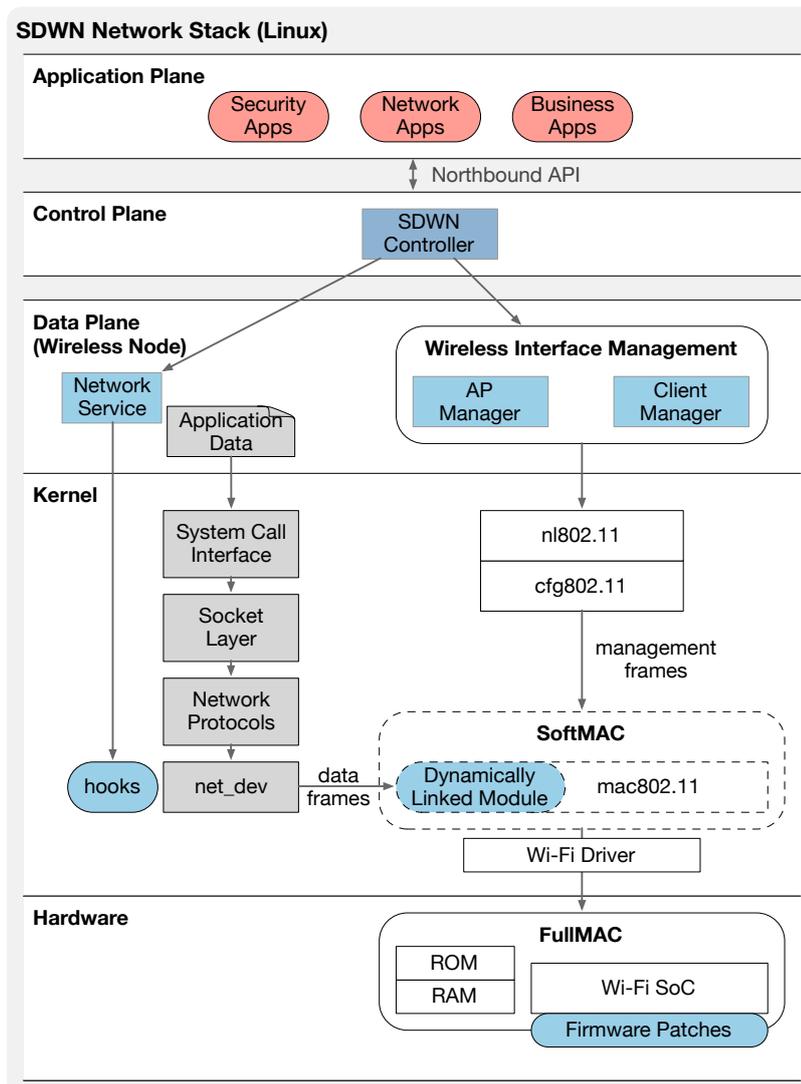
Recently, Software-defined Networking (SDN) also found its way from wired into wireless networks. Software-defined Wireless Networks (SDWNs) introduce datapath programmability, fine-grained transmission control, and programmable wireless virtual network functions used in Wi-Fi access points. In contrast to wired SDN networks, SDWN have to handle several peculiarities due to the communication through a shared medium with quickly changing characteristics and due to its feature-driven and complex PHY- and MAC-layer design [Sch16].

According to the Institute of Electrical and Electronics Engineers (IEEE), Wireless Local Area Networks (WLANs) and their PHY- and MAC-layer properties are specified within the IEEE 802.11 standard, which is part of the IEEE 802 family [Gas05].

In Figure 2.3, the IEEE 802 family specifications are shown. It focuses on the two layers at the bottom of the OSI reference model. On one hand, wired IEEE standards are shown. For example, 802.1 provides management features such as bridging and virtual LANs. On the other hand, 802.3 is the specification of Carrier Sense Multiple Access with Collision Detection (CSMA/CD), which is used as a mechanism for medium access control in Ethernet LANs. 802.5 specifies the Token Ring LAN. Furthermore, 802.2 specifies the logical link control (LLC), an upper sublayer within the data link layer providing multiplexing and coexistence of network layer protocols within the same network medium.

On the other hand, IEEE 802.11 specifies the MAC and two PHY layer protocols for Wireless LANs (WLAN). 802.11a, 802.11b and 802.11g represent different revisions at the physical layer, while 802.11n introduces major revisions to the *MAC* and *PHY* layer.

Figure 2.4 shows SDWN network stack including programmable components at different layers: at application layer, within the operating system kernel and on a FullMac chip. At the top,



**Figure 2.4** Software-defined Wireless Networking including programmable components at application layer, within the operating system kernel and on a FullMac chip.

network applications (application plane) instrument the SDWN controller (control plane) via its northbound API. The SDWN controller instructs the data plane, i.e. it controls the wireless interface management processes (right) and network services (left) on a wireless node. A wireless node can be part of basic service sets that operate in two different modes: infrastructure mode controlled by a wireless access point (AP), and ad-hoc mode formed by independent peers.

The data path for incoming packets is shown in the middle of the data plane (gray). At the top, applications that are running as processes on a wireless node use system calls to access the socket layer of the operating system (OS). Within the OS kernel, network protocols transform the application data to frames that are going to be transmitted via the MAC layer.

Basically, the 802.11 data path is divided into upper- and lower-level MAC. The upper-level MAC is responsible for packet state handling for those parts of the MAC protocol that are not time

critical. The lower-level MAC interfaces directly to the PHY transmission path and handles all wireless transmissions and receptions, particularly all time critical aspects [SWH16]. On the right of Figure 2.4, SoftMAC and FullMAC realizations of the MAC layer are shown. While SoftMAC realizes the upper-level MAC in software, the FullMAC implementation realizes both MAC-levels in hardware on a dedicated processor on the Wi-Fi System-on-a-Chip (SoC). Modifications to the MAC layer at runtime can be made by linking a kernel module dynamically, or by patching the Wi-Fi firmware dynamically. Network Services and wireless interface management processes like the AP manager running on access points and the client manager running on clients control the association and authentication of wireless devices (nl802.11 and cfg802.11) as well as packet flow rules that can be specified with hooks inside the kernel.

## 2.2 Virtualization

The term virtualization covers a broad variety of resource-management technologies. Each of them creates an abstraction of a physical resource that hides management details from users and applications. Virtualization allows users and applications to operate on a virtual version of, e.g., servers, data storage, or networks, instead of struggling with management details.

### 2.2.1 Virtual Machines

A common definition of virtualization was given by Popek and Goldberg [PG74] in 1974. They describe a virtual machine (VM) as an efficient, isolated duplicate of a real machine created by a piece of software that is in complete control of the system resources: a virtual machine monitor (VMM). It therefore realizes three essential characteristics: equivalence, efficiency, and resource control.

“

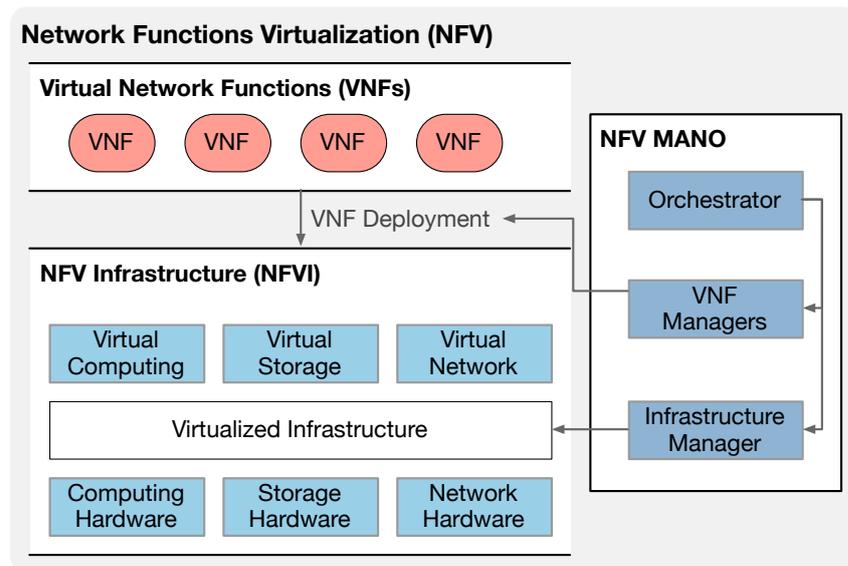
1. **Equivalence:** Any program run under the VMM should exhibit an effect identical with that demonstrated if the program had been run on the original machine directly, with the possible exception of differences caused by the availability of system resources and differences caused by timing dependencies.
2. **Efficiency:** It demands that a statistically dominant subset of the virtual processor's instructions be executed directly by the real processor, with no software intervention by the VMM. This statement rules out traditional emulators and complete software interpreters (simulators) from the virtual machine umbrella.

**3. Resource Control:** It labels as resources the usual items such as memory, peripherals, and the like, although not necessarily processor activity. The VMM is said to have complete control of these resources if (1) it is not possible for a program running under it in the created environment to access any resource not explicitly allocated to it, and (2) it is possible under certain circumstances for the VMM to regain control of resources already allocated.

*Popek/Goldberg: Formal Requirements for Virtualizable 3rd Gen. Arch. [PG74]* ”

Traditionally, server operators were forced to run only one operating system at a time on a single server. In contrast, virtualization enables operators to run multiple instances of one or more operating systems simultaneously in virtual machines on a single server. Since virtual machines can be described as a set of configuration and virtual storage files, they can be deployed much more easily and quicker than physical servers. Furthermore, virtualization is not limited to creating virtual machines. It can be applied at different levels:

- 1. Hardware or Platform Virtualization** enables the user to run custom guest operating systems in virtual machines. There are two types of platform virtualization: Para-virtualization and full virtualization. The concept of full virtualization is based on the emulation of hardware components. While network-cards and other I/O-devices are fully emulated, accesses on important components such as CPU or memory are passed-through and executed natively for performance reasons. That is why full virtualization performance is close to the performance of native execution apart from I/O-intensive applications. The main advantage of full virtualization is that no modifications in the guest operating system are needed, in contrast to para-virtualization, where all hardware components are accessed directly via hypercalls, a special hypervisor interface.
- 2. Operating System Virtualization** enables the operating system to place processes in mutually isolated runtime-/system-environments. For example, this concept is implemented through jails in BSD operating systems [MNW14] and Linux Containers (LXC), based on Linux namespaces, cgroups, chroot. These virtualized environments can be provisioned very fast (in milliseconds), and the underlying hardware need not be virtualized, which reduces the virtualization overhead significantly. On the other hand, it is not possible to run different guest operating systems inside these virtualized environments.
- 3. Application Virtualization** describes a virtualization type on application layer, i.e. applications are compiled into a special type of virtual machine code and executed in a virtual machine application.



**Figure 2.5** The approach of Network Function Virtualization (NFV) according to the reference architecture of the ISG [Hu+15].

## 2.2.2 Network Function Virtualization

Network Functions Virtualization (NFV) is defined as the virtualization of network functions from proprietary hardware platforms by implementing these functions in software and running them on virtual machines. In particular, network function virtualization focuses network function devices (switches, routers, access points, deep packet inspectors), network-related compute devices (firewalls, network management systems) and network-attached storage (network file and database servers) [Sta15].

Figure 2.5 describes the approach of NFV according to the reference architecture of the ISG [Hu+15]. The objectives of NFV are to consolidate many network equipment types onto commodity server hardware that can be located in data centers and network nodes. Therefore, a NFV MANO framework orchestrates and manages the resources in the NFV environment, consisting of the (i) NFV infrastructure (NFVI) providing virtualized computing, storage and network resources on top of physical hardware, and (ii) the Virtual Network Functions (VNFs) that are deployed within the NFVI.

## 2.3 Computing Paradigms

In the following, the computing paradigms that are related to this thesis are described: cloud computing, fog computing, mobile edge, and mobile cloud computing.

### 2.3.1 Cloud Computing

Cloud computing realizes the service provisioning model of utility computing [Par66], that has already been discussed since the 1950's. According to the utility computing paradigm, computing should be publicly available as needed in a pay-as-you-go manner, comparable to water from the tap or electricity from the power grid. The name of the *grid computing* paradigm also referred to this analogy. Grid computing suggested protocols offering federated computation and storage resources for a specific community. By contrast, *cloud computing* provides computing resources and services via Internet, accessible for all kinds of customers [Bau+10]. The term *cloud* in the name of this paradigm refers to the icon of a cloud that is commonly used as a symbol for the Internet.

Besides utility computing, the central ideas of cloud computing are Virtualization (described in more detail in Section 2.2), Service-Oriented Architecture (SOA) and Service Level Agreements (SLA) [Bau+10]. SOA is an architecture that consists of a collection of loosely coupled, platform-independent, and distributed services that are accessed via Web Service protocols. Commonly used Web Service protocols are the SOAP protocol that is based on XML remote procedure calls and the REpresentational State Transfer (REST) protocol that is based on stateless communication via HTTP methods. A SLA, however, specifies Quality-of-Service (QoS) metrics like availability, throughput, latency, etc. that are subject to a contract between a network service provider and a customer. Such a contract guarantees a certain level of QoS and defines penalties for failures to comply.

In addition to these central ideas, cloud computing has certain characteristics. Most important is the property of being scalable and elastic, i.e., being able to handle virtually unlimited amounts of work (scalability), and to add or remove resources on a short notice (elasticity). Both properties are achieved with dynamic provisioning of computing instances and a multi-tenant design, i.e., the software-controlled deployment and removal of computing instances like virtual servers (dynamic provisioning) and the partitioning of data, configuration, as well as network- and computing resources, so that clients are virtually isolated from each other (multi-tenancy) [Bau+10]. Furthermore, cloud computing provides three service models and four deployment models. The service models are defined as follows [Bau+10]:

**Infrastructure-as-a-Service (IaaS):** The IaaS service model provides an interface to computing resources like processing, storage, network etc. For example, the Amazon Elastic Compute Cloud (Amazon EC2)<sup>3</sup> provides low level control of computing resources. In addition of physical resources, the customer typically controls virtualized resources, i.e., logical entities that are abstracted from the underlying physical entities.

<sup>3</sup><https://aws.amazon.com>

**Platform-as-a-Service (PaaS):** In contrast to IaaS, the PaaS service model provides a programming and execution environment where the customer can develop and run software using a provider-defined programming language. Google AppEngine<sup>4</sup>, for example, is targeted exclusively at traditional web applications and enforcing an application structure of clean separation between a stateless computation tier and a stateful storage tier allowing automatic scaling and high-availability mechanisms.

**Software-as-a-Service (SaaS):** SaaS service model refers to software applications and services that run in a cloud computing infrastructure, on top of PaaS or IaaS resources, and directly address the consumer. Full-fledged applications are typically accessed uniformly via a Web browser from a thin client.

The four deployment models in cloud computing essentially describe in which administrative domain cloud systems are deployed and to whom cloud services are made available [Bau+10]:

**Public Cloud:** This deployment model makes services available to the general public, i.e., service providers and consumers are not part of the same organization.

**Private Cloud:** In contrast to public clouds, private clouds provide services to the users within a single organization. The underlying physical resources as well as user data are not shared with or transferred to third parties. Nevertheless, cloud infrastructure software is used to provide clean service interfaces to the members of this organization.

**Community Cloud:** A community cloud provides services for exclusive use by a specific community that has shared concerns.

**Hybrid Cloud:** Hybrid clouds are used in scenarios where both public and private cloud services are used. Typically, public cloud resources are provisioned to meet seasonal demand variations or burst demands for an exogenous event.

### 2.3.2 Fog Computing

According to the OpenFog consortium, the cloud computing paradigm alone is unable to meet the recent challenges like Internet of Things (IoT), i.e., the extension of Internet connectivity to any physical object and enhanced device- and network-level capabilities for the next generation mobile networks (5G). The core characteristic of fog computing is that it provides computing services closer to the user and to the devices that generate the input data for computations.

“ Fog computing is a system-level horizontal architecture that distributes resources and services of computing, storage, control, and networking anywhere along the continuum from Cloud to Things, thereby accelerating the velocity of decision making. Fog-centric architecture serves a specific subset of business problems that can not be successfully implemented using only traditional cloud based architectures or solely intelligent edge devices.

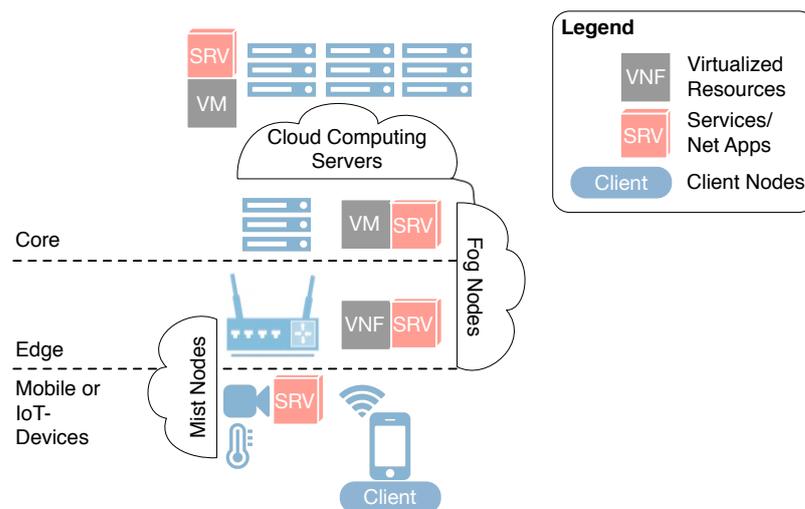
*OpenFog Consortium [Gro17]* ”

---

<sup>4</sup><https://cloud.google.com/appengine>

A fog architecture consists of several tiers of fog nodes. Fog nodes are the core components in a fog computing architecture and are either physical or virtual components. They can be organized either in vertical clusters to support isolation, or in horizontal clusters to support federation. Nodes at the edge are typically focused on data that is produced at the edge, i.e., on mobile devices, sensors, or actuators. Nodes at the next tier filter and transform data, but also perform latency-critical analytics. Nodes at the higher tiers, close to fog nodes on cloud instances, are typically used to turn data into knowledge [Gro17].

In contrast to the cloud computing paradigm, fog computing (i) offers low latency due to the fog nodes' contextual location awareness, (ii) demands widely, geographically-identifiable distributed deployments, (iii) supports seamless data transfer for certain services due to inter-operating fog nodes and federated services, (iv) involves real-time interactions rather than batch processing [Ior+18]. In extension to cloud computing, microservices are predominantly used as a specialization of SOAs. Microservices, or lightweight services, enable a more fine-grained application structure. It aims to improve modularity, manageability and incremental development of software [Gro17]. Moreover, according to NIST, three fog node architectural service models (IaaS, PaaS, and SaaS) and four fog node deployment models (public, private, community, hybrid) are defined [Ior+18]. Furthermore, mist computing is defined by NIST as an additional layer leveraging more specialized and dedicated nodes at the edge of the network that exhibit low computational resources [Ior+18].

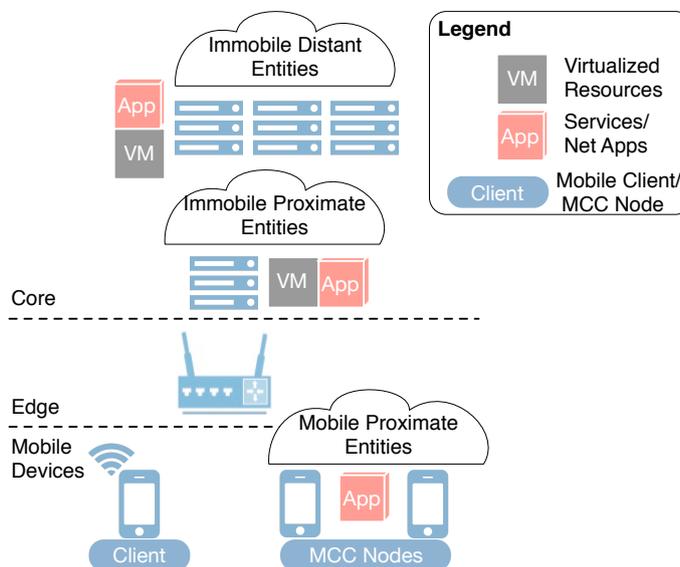


**Figure 2.6** The fog computing paradigm.

In Figure 2.6, the fog computing paradigm is depicted. At the top, cloud computing servers still provide (virtualized) resources, services and network applications (net apps). Cloud services are located at the core of the network in centralized data centers. Fog nodes, however, can be (i) different kinds of server clusters like cloudlets/micro-datacenters close to the edge or (ii) edge devices like wireless access points/routers. They provide resources for services and net apps, although near-edge server clusters typically have more capacity than edge-devices to run storage- and compute-intensive services/net apps. Therefore, Figure 2.6 depicts a virtual network function (VNF) as virtualized resource of the edge device. Furthermore, mist nodes are either IoT-devices like sensors/actuators or edge-devices like switches/routers. They perform lightweight operations directly on sensors/actuators or other components of the network fabric, particularly within industrial applications.

### 2.3.3 Mobile Cloud Computing

Fog computing extends the cloud to the edge of the network. Nevertheless, another edge-oriented paradigm emerged in parallel to fog computing: Mobile Cloud Computing (MCC). This paradigm integrated mobile computing into the cloud computing paradigm by leveraging scalable cloud resources for mobile devices. In this paradigm, there is a distinction between (i) distant and proximate computing entities as well as (ii) mobile and immobile entities. On the one hand, cloud computing providers like Amazon EC2 provide large-scale pools of centralized computing resources. In mobile cloud computing, these distant immobile computing entities are supplemented by proximate immobile computing entities. Fixed virtualization servers like cloudlets [Sat+09], i.e., micro cloud-datacenters close to the edge of the network, were used to offload computationally intensive tasks to mobile cloud services. On the other hand, proximate mobile computing entities, i.e., ad-hoc connected groups of mobile devices, were used to provide additional offloading resources [Abo+14].

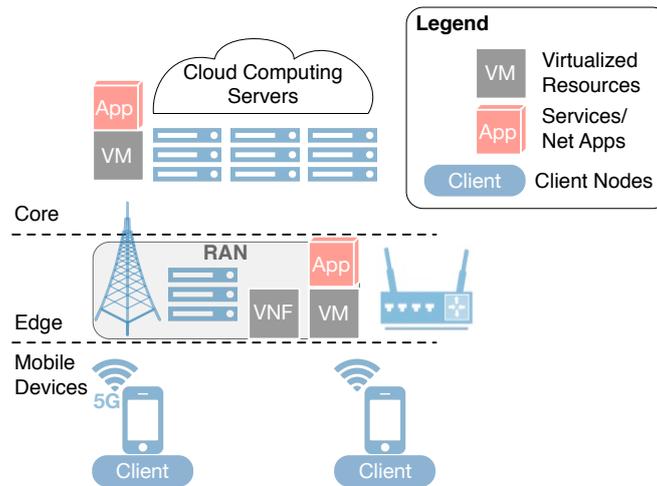


**Figure 2.7** The mobile cloud computing paradigm.

In Figure 2.7, the mobile cloud computing paradigm is depicted. At the top, immobile distant entities represent public cloud resources. Furthermore, computational and storage intensive tasks can be offloaded from mobile phones to immobile proximate entities. They are located close to edge devices, but are not integrated into the network of the telecommunication or Internet provider. Moreover, services and network applications can be offloaded to nearby MCC nodes. They consist of mobile devices forming an ad-hoc network. In contrast to virtualized server, they do not provide access to the software stack from the kernel upwards. Instead, an API for processing tasks is typically offered at application level.

### 2.3.4 Mobile Edge Computing

In mobile edge computing (MEC), cloud computing services are extended to edge devices in proximity to the end user. In Figure 2.8, the mobile edge computing paradigm is depicted. Cloud resources like storage and processing capabilities are deployed within the Radio Access Network



**Figure 2.8** The mobile edge computing paradigm.

(RAN) of cellular network base stations, providing mobility, location and context awareness support [Abb+18].

Mobile edge computing addresses several next-generation mobile network challenges: low latency, proximity, high bandwidth, and real-time insight into radio network information and location awareness. Novel technologies such as Software-defined Networks (SDN) and Network Function Virtualization (NFV), which are both discussed in detail in Section 2.1, also feature network programmability and virtualized network components instead of custom network hardware appliances. According to the European Telecommunications Standards Institute (ETSI), mobile edge computing is a key emerging technology for 5G networks, next to SDN and NFV.

The relation between mobile edge computing and these emerging technologies is described as follows:

“ In fact, while NFV is focused on network functions, the MEC framework enables applications running at the edge of the network. The infrastructure that hosts MEC and NFV or network functions is quite similar; thus, in order to allow operators to benefit as much as possible from their investment, it will be beneficial to reuse the infrastructure and infrastructure management of NFV to the largest extent possible, by hosting both VNFs (Virtual Network Functions) and MEC applications on the same platform.

*ETSI White Paper [Hu+15]* ”

### 2.3.5 Comparison of Computing Paradigms

To sum up, fog, mobile cloud, and mobile edge computing reflect a recent shift from the core to the edge of the network: more and more applications and services make heavy use of information provided by mobile devices, Internet-connected "things", sensors and actuators. On the one hand, these applications rely on some sort of computing capability close to the source, to gather and aggregate local data before sending them to a remote server. On the other hand, proximate

**Table 2.1** Features of Different Computing Paradigms  
Features of different computing paradigms according to [RLM18].

	Cloud	Mobile Edge	Mobile Cloud	Fog	Mist
<b>Ownership</b>	IT companies	Telco companies	Private entities, individuals		
<b>Deployment</b>	Network core	Network edge	Near-edge, user devices	Near-edge, edge	IoT devices
<b>Hardware</b>	Servers	Base stations, servers	Servers, mobile devices	Servers, access points, routers	Micro-controllers
<b>Power Consumption</b>	Megawatts	Kilowatts	Watts/ Kilowatts	Watts/ Kilowatts	Milliwatts
<b>Technology</b>	VMs, SDN, NFV	VMs, SDWN, NFV	VMs, ad-hoc networks	VMs, SDN, NFV	IoT OS
<b>Network Architecture</b>	Central	N-tier, decentralized, distributed			
<b>Mobility</b>	N/A	Yes			
<b>Latency</b>	Average	Low			
<b>Local Awareness</b>	N/A	Yes			

client devices are increasingly exchanging information with each other and often access cloud computing resources that can be cached locally.

Table 2.1 shows the features of different computing paradigms. Within the network core, data centers operated by private companies/organizations are the infrastructural basis for cloud computing services. These data centers provide large pools of storage as well as server capacities and provide services on top of a virtualized infrastructure. In terms of power consumption, data centers consume megawatts of power for computing servers and the required cooling infrastructure. The centralized approach of cloud computing allows savings in costs gained by an increased number of computing instances, but cloud services rely on Internet communication over large distances. They are neither optimized to low latency, to the mobility of the end users, nor are APIs provided to access local information like network statistics or sensor data.

Fog computing was intended to complement classic cloud computing by providing computation, network, and storage services in between the end devices and the centralized cloud infrastructure. In today's definition, fog nodes can include resource-constrained devices, mobile end devices, and Internet routers [Ior+18]. In fact, fog computing constitutes a multi-tier architecture consisting of clients, fog nodes, and central servers that cooperate and coordinate each other. Fog nodes are context-aware and support a common data management and communication system.

Mobile edge computing provides cloud computing capabilities on mobile base stations at the network's edge, including low latency, access to wireless mobile telecommunication network information, and location awareness. Mobile edge computing environments consist of virtualization servers that are deployed at LTE/5G base stations, and that allow related services like Network Function Virtualization (NFV) and Software-defined Networking (SDN) [Hu+15].

The focus of mobile cloud computing is on delegating compute- and storage-intensive tasks from resource-constrained mobile devices to cloud computing services. While tasks were offloaded

into the centralized cloud in early versions of this concept, today both fixed virtualization servers close to the network edge and mobile computing entities are leveraged to provide offloading capabilities.

## 2.4 Energy Efficiency

In the following, definitions of energy efficiency are given. Furthermore, power management techniques for CPUs, in data centers and for mobile devices are described.

### 2.4.1 Definitions

According to the International System of Units (SI), the unit the power consumption of a system is measured is Joule per Second (J/s) or Watt (W) [TT08]. The energy consumed by a computing system in the time interval  $[t_0, t_1]$  is defined as the integral of power consumed in the domain of integration, shown in Equation (2.1).

$$E(t) := \int_{t_0}^{t_1} P(t) dt. \quad (2.1)$$

Energy is measured in the SI unit Joule (J), 1 Joule being defined as the amount of energy needed to continuously produce one Watt for one second, i.e. 1 Joule = 1 Wattsecond (Ws). For electrical energy, the measurement unit is typically upscaled to Watt-hours (Wh).

Energy efficiency of a computing system is defined as the relation of energy consumed and useful work done, shown in Equation (2.2). A computing system is called energy-efficient if it is able to perform computations without wasting energy.

$$\text{Energy Efficiency} = \frac{\text{Useful Work}}{\text{Total Energy}} \quad (2.2)$$

The denominator *Useful Work* in the fraction refers to a metric that is specific to the problem domain. Therefore, Equation (2.2) can take specific forms for different areas: In the area of High-Performance Computing (HPC), for example, it is common to compare the energy efficiency of supercomputers by measuring the Floating Point Operations Per Second (FLOPS) per Watt during numerical linear algebra computations<sup>5</sup>.

In industrial data centers built from standard server hardware, server-level benchmarks like *SPEC\_power* are used to measure the ratio between performance and power (Performance per Watt)<sup>6</sup>. According to their methodology, all power-measured benchmarks have at least two distinct measurement segments: measuring the power used while generating the maximum performance metric (*full performance*) and the power used during periods when the system is ready to do work, but has not done work for a period of time (*active idle*). In addition, *SPEC\_power* benchmarks the CPUs, caches, memory hierarchy, and the scalability of shared memory processors for multiple levels of synthetic system loads<sup>7</sup>.

<sup>5</sup><https://www.top500.org/green500>

<sup>6</sup><https://www.spec.org/power>

<sup>7</sup>[http://www.spec.org/power/docs/SPEC-Power\\_and\\_Performance\\_Methodology.pdf](http://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf)

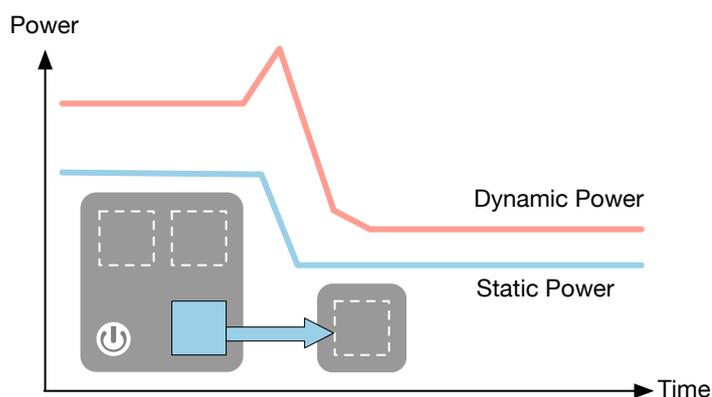


Figure 2.9 Task migration within asymmetric multiprocessors.

## 2.4.2 CPU Power Management

Related terms to active idle and full performance are *static power* and *dynamic power consumption*. These terms are used by CPU designers to characterize the influence of different parameters to the power consumption of a processor. Static power is the amount of power that is consumed without performing useful work. It is caused by transistor leakage currents and depends on the temperature of a system. Dynamic power is the amount of power that is consumed by active logic gates inside a processor, i.e., when useful work is performed. It depends on the processor frequency.

Equation (2.3) defines the CPU dynamic power consumption. Both power consumption and operating temperature of a device are proportional to the capacitance  $C$ , the operating voltage  $V$  and the switching frequency  $f$ .

$$P = C * V^2 * f \quad (2.3)$$

Dynamic Voltage and Frequency Scaling (DVFS) is a common power management technique to dynamically reduce or increase the supply voltage and the operating frequency of a CPU. It is used to adapt its dynamic power consumption to its utilization, to conserve power and to reduce the dissipation of heat. DVFS is a commonly used technique in CPUs, especially in mobile CPUs, and typically complemented with power states that aim to reduce the static power of a system, such as sleep states accessible via the Advanced Configuration and Power Interface (ACPI)<sup>8</sup>.

Beside symmetric multiprocessor architectures, asymmetric multiprocessors (AMP) have gained some interest of mobile chip designers recently. For example, a widespread ARM-based Samsung Exynos5422 asymmetric multicore processor (AMP) consists of two cores: a big but power-hungry A15 and a little, low-power A7 processor, which share the same instruction set but have a significantly different architecture. Regarding power-efficiency, the little processor has a 40% better performance/energy ratio, while the Cortex A15 processor is able to process 30% more load. Task scheduling and -migration between different cores are used to adapt the power consumption to the requirements of the applications executed on the processor [Mit16].

Figure 2.9 depicts a task migration within a asymmetric multiprocessor. Initially, a long-term task is executed on the big core with a better performance/power ratio but with a higher static

<sup>8</sup><http://www.acpi.info>

power consumption than the little core. In order to reduce the CPU's power consumption, the task is migrated from the big core to the little core. A task migration can take millions of CPU cycles for flushing and warming up caches [Mit16]. It might therefore result in a short dynamic power consumption peak, as depicted in Figure 2.9. After the task migration is finished, the big core can be put into sleep mode and the total power consumption drops significantly.

### 2.4.3 Data Center Power Management

A data center is a building delivering utilities needed to house interconnected computing and storage infrastructure: power, cooling, shelter, and security. Most data center are smaller than 450m<sup>2</sup> with less than 1 MW of power consumed by IT equipment, but large data center support 10-30 MW of IT equipment power [BCH13].

Power and Energy management in data centers is a key issue for IT equipment operations, focusing on the reduction of all energy-related costs. To measure energy efficiency in a data center, several metrics have been defined. Equation (2.4) shows the definition of energy efficiency for data centers according to Barroso and Hoelzle [BCH13].

$$\text{Efficiency} = \underbrace{\frac{1}{PUE}}_{(a)} * \underbrace{\frac{1}{SPUE}}_{(b)} * \underbrace{\frac{\text{Computation}}{\text{Total Energy to Electronic Components}}}_{(c)} \quad (2.4)$$

The first term  $(a)$  in Equation (2.4) reflects the quality of the data center building infrastructure, the ratio of total building power to IT power, called Power Usage Effectiveness (PUE). It is expressed in Equation (2.5).

$$PUE = \frac{\text{Facility Power}}{\text{IT Equipment Power}} \quad (2.5)$$

The PUE metric is widespread used and reported by data center operators. For example, the Facebook company spent 1.8 million MWh on their data centers in 2016, each operating with a Power Usage Effectiveness (PUE) between 1.06 and 1.1<sup>9</sup>.

The second term  $(b)$  in Equation (2.4) reflects the overhead inside servers or other IT equipment, called Server PUE (SPUE). This is expressed in Equation (2.6).

$$SPUE = \frac{\text{Total Server Input Power}}{\text{Useful Power}} \quad (2.6)$$

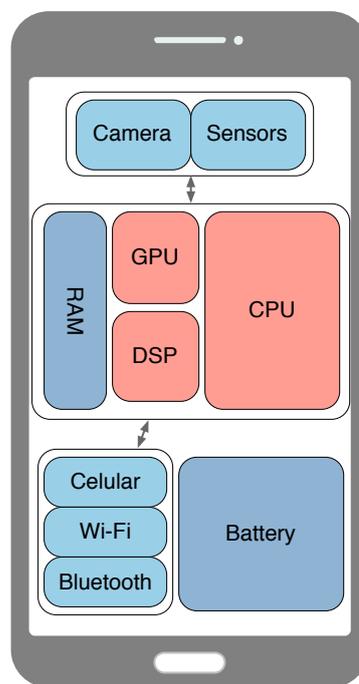
The term Useful Power includes only the components directly involved in the computation, like CPUs, memory, I/O components, but not the power supply module and the cooling fans.

The product of both  $(a)$  and  $(b)$  reflects the electromechanical efficiency, called True PUE ( $TPUE = PUE * SPUE$ ).  $(c)$  accounts for how the electricity delivered to electronic components is actually translated into useful work [BCH13].

<sup>9</sup><http://sustainability.fb.com/our-footprint>

One of the major downsides of power management techniques are energy and latency penalties after waking up from hardware sleep states. In the recent past though, Service Level Agreements (SLAs) have become an integral part of many virtualized data centers. With respect to power and energy management, tradeoffs between SLA violations and improved energy efficiency need to be taken into account. This has been addressed, for example, by [Voo+09; BB10a; Bor+12]. In their solutions, models for performance degradation during active power management were developed and used for reactive and proactive strategies for placing and relocating Virtual Machines (VMs).

### 2.4.4 Mobile Phone Power Management



**Figure 2.10** Overview over smartphone subsystems.

Figure 2.10 gives an overview over the components of a smartphone System-on-a-Chip (SoC). Typically, these SoCs contain a mobile ARM CPU with RISC architecture. In the center, CPU, a Graphics Processing Unit (GPU), and a Digital Signal Processor (DSP) are shown. While a CPU can be used for general tasks like Web applications, the GPU and the DSP are optimized special-purpose processing cores: a GPU for parallel processing of images and arrays, DSPs can be used to offload tasks. For example, transforming pictures taken from the internal cameras or filtering sensor values can be offloaded from the CPU to the DSP. Since mobile phones are powered by batteries that have a limited capacity, power management for mobile devices is crucial.

Energy management techniques on mobile phones can be distinguished between (i) adjusting power states of processing units, and (ii) leveraging other computing resources, like offloading to a server, to another mobile device or to another component on that device [KKC18].

Furthermore, Figure 2.10 shows cellular, Wi-Fi and Bluetooth wireless network interfaces. In a wireless network, signals are transmitted in the form of electromagnetic waves. The signal is attenuated during the transmission and arrives at the receiver with a reduced power. This weakening is influenced by several factors: (i) Path losses are caused by attenuation of the power of the radiated signal as well as influences of the propagation path. The signal is influenced, for example, by the distance between transmitter and receiver. (ii) Shadowing is caused by barriers between the transmitter and receiver, such as buildings, vehicles and walls, which attenuate transmission power through absorption, reflection, scattering and diffraction. If the damping is too high, this can lead to signal loss. (iii) Fading refers to signal fluctuations on the receiver side due to overlapping effects caused by interference.

A radio's transmit and receive power is typically measured in the power ratio expressed in decibels (dBm) with reference to 1 mW. For example, a typical cellular phone call consumes 27 dBm or 500 mW, a typical Wi-Fi transmission 15 dBm or 32 mW. On the receiver's side, the Received Signal Strength Indicator (RSSI) is an important metric to measure the power present in a received signal [Tar+14].

In 802.11 Wi-Fi networks in infrastructure mode, power management allows clients to enter a power save mode (PS). A central access point (AP) buffers packets for hosts in PS mode. During PS mode, clients turn their antennae off and wake up periodically to retrieve buffered packets from the AP. Furthermore, wireless networks such as GSM or 3G networks have different modes of operation: (i) the standard mode *Idle*, (ii) a mode for higher throughput and lower latencies *DCH (Dedicated Channel)*, (iii) a mode for lower throughputs *FACH (Forward Access Channel)*. The time span between the last data transaction and the change from DCH to FACH is called *tail time*. The *tail energy* describes the energy that accumulates in the tail time. The timeouts are set by the network operator and vary depending on the operator or network type [MKC12; MNR13].

## 2.5 Summary

This chapter introduced fundamental terms, concepts, and technologies used in this thesis. Section 2.1 explained communication networks. Section 2.2 described the concept of virtualization. In Section 2.3 the computing paradigms of cloud, fog, mobile cloud and mobile edge computing were introduced. The definition of energy efficiency was given and power management for data centers and mobile phones were described in Section 2.4.



# 3

## Transitional Near-\* Computing

In this chapter, the novel concept of transitional near-\* computing is presented. First, near-\* computing is defined in Section 3.1. In Section 3.2, the basis of transitional computing is described: mechanism transitions in communication systems. Transitional computing shifts the scope of mechanism transitions from communication systems to computing systems. A more detailed definition is given in Section 3.3. Section 3.4 describes the application of transitional computing to improve the energy efficiency of near-\* computing environments. Finally, Section 3.5 summarizes the chapter.

### 3.1 Near-\* Computing

According to the Association of Computing Machinery (ACM), the term *computing* is defined very generically as *to process, to structure, to find and to gather various kinds of information* [Sha+06]. It applies to many modern computing platforms, for example, to warehouse-scale computers [BCH13] that are used for computing paradigms such as cluster, grid and cloud computing [Fos+09]. More recently, fog computing [Bon+12] and edge computing [OBL15] emerged as novel computing paradigms. In the following, the approach to perform computations near-user, near-network, near-device, and near-data is described in the context of these recent paradigms. Afterwards, near-\* computing is defined.

#### 3.1.1 Near-User

In the recent past, several computing paradigms were proposed to perform computations in proximity to the end users and mobile devices, e.g., fog computing [Bon+12], mobile cloud computing [Hoa+13], and mobile edge computing [OBL15]. Fog computing stems from data center and network operators. Its purpose is to extend cloud paradigms towards the network and closer to the network's edge. Mobile cloud computing, on the other hand, has its roots in peer-to-peer computing. It uses network communication on the application layer, but is not integrated into a software-defined network. Furthermore, (multi-access) edge computing originates from the telecommunication sector. It was originally aimed at realizing computation capabilities close to or within a cellular base station, but as of today, edge computing also includes resources of all edge devices that allow end users to access the network, e.g., Wi-Fi access points. These three paradigms do not share a common terminology, since they originate from different players. But they share software technologies like virtualization and software-defined networks, that are used within these paradigms to improve network latency, reduce data traffic, and to leverage local context information by performing computations *near-user*, i.e., geographically close to the end user.

### 3.1.2 Near-Network

Fog and edge computing already leverage technologies like software-defined networks and virtualized network functions that enable fine-tuned orchestration of computing instances and network communication. Radio Access Networks (RANs) in edge environments, for example, allow collaborative radio access at the management layer and real-time information about the network at the application layer. Furthermore, Software-defined Wireless Networks (SDWNs) in fog environments introduce datapath programmability, fine-grained transmission control, and programmable wireless virtual network functions used in fog/edge Wi-Fi access points. These technologies allow computations that are closely integrated into the network to be performed *near-network*, i.e., in proximity to network components and with awareness of the network's state.

### 3.1.3 Near-Device

Current computing paradigms like fog and mist computing [Ior+18] focus on providing computing resources for the increasing number of Internet-connected machines and objects ("things"). By 2020, about 50 billion of this kind of devices are expected [Eva11]. According to these paradigms, computations should be performed either directly within the network fabric at the edge of the network or *near-device* embedded in the end devices themselves. These heterogeneous microcontroller-based end devices have a limited computing- and battery power, which reduces the number and type of possible embedded applications. Nevertheless, they are typically able to perform diverse data transformations. These can be used to reduce the number of data transfers that are expensive in terms of the transmission power consumed.

### 3.1.4 Near-Data

Two trends became visible in recent years in the operating system and networking research communities. Both are aimed at increasing network- and storage throughput and decreasing the latency of I/O operations. First, datapaths from I/O devices to applications are streamlined by reducing the overhead of the operating system (e.g., [Pet+16; Bel+17]). Second, datapaths are offloaded to fixed-function processors with limited support for generic computations. For example, "smart" flash storage devices (SmartSSDs) and network interface cards (SmartNICs) support programmable query- and packet-processing planes [Do+13; Sil17]. In these devices, I/O processing is performed on a separate processor without the involvement of the main CPU, gaining additional efficiency with respect to latency, but also to power consumption. These trends can be summarized as high throughput and low latency *near-data* processing, i.e., to perform computations close to I/O components.

### 3.1.5 Definition of Near-\* Computing

Near-user, near-network, near-device, and near-data computing are terms reflecting four current trends pushed forward by different research communities and industry players. Combined, they form a novel perspective on current computing paradigms summarized as *near-\* computing*.

**Near-\* Computing** A novel perspective on current computing paradigms summarizing four major trends that focus on the locality of computations in heterogeneous platforms with a multitude of functions that are connected to a continuum of computational resources.

## 3.2 Multi-Mechanism Adaptation

The concept of mechanism transitions in a communication system was developed and formalized by the Collaborative Research Center **MAKI (Multi-Mechanism Adaptation for the Future Internet)** [Frö+16; Frö+15; Ste+15; RKS15] funded by the German National Science Foundation. The challenges that are addressed by this concept already play an important role in the current Internet:

“

1. The dramatically increasing dynamics of the Internet leads to a quest for dynamic switching between protocols and other schemes, while
2. only few, rather simple examples of corresponding approaches exist today and
3. little knowledge exists about the relationship between ‘conditions’ (e. g., network load, signal strengths) and performance of individual protocols or communication schemes while
4. crosslayer optimization between several dynamically adaptable layers is far from being common in the Internet.

*Mechanism Transitions [Frö+16], pp. 6* ”

Several domain-specific protocols already adapt their behavior at runtime in order to react to dynamically changing network conditions. Although some of today’s protocol- and application-specific solutions already implement adaptive approaches, the Internet deviates from clean models required by a concept that switches between exchangeable mechanisms. By contrast, mechanism transitions are thought of as a fundamental principle for a more flexible and dynamically adaptable future Internet. MAKI’s research approach considers flexible and exchangeable components as a generic key contribution to a future Internet, where the components of a networked system can be adapted at runtime.

### 3.2.1 Mechanism Transition

In the Internet, the rules of interaction between interconnected systems are standardized by the ISO reference model for Open System Interconnection (OSI) [Zim80]. Its architecture is logically composed of a stack of seven layers, where each layer wraps the lower layers and isolates them from the higher layers, and where each layer provides specific services and protocols. By conforming to the OSI reference model, a system will be open to communicate with all other systems worldwide implementing the same standards, independent of their internal organization and functioning.

In analogy to services and corresponding protocols in the OSI model, *mechanisms* are abstract concepts that are realized by cooperating runtime modules. But in contrast to the rigid structuring into layers, protocols, and services in the OSI model, the concept of a mechanism is more universal.

**Mechanism** A mechanism is a conceptual element of a networked system that is bound to a realization as cooperating functional units [Frö+16].

Mechanisms can include (i) functionality that corresponds to a service in the OSI sense, like a reliable data transfer provided by a transport layer service and a corresponding protocol (e.g., the TCP protocol), (ii) functional aspects of a protocol like flow and congestion control, or (iii) conceptually confined functionality of a middleware or of an overlay. Mechanisms consist of distributed mechanism entities that cooperatively provide the mechanism. They are interconnected through logical or physical connections, and this connectivity meshwork is called a mechanism topology.

In mechanism transitions, equivalent or similar mechanisms are by design inherently exchangeable. This exchange of mechanisms is called a mechanism transition.

**Mechanism Transition** A mechanism transition is the functional replacement of a source mechanism by a functionally similar or equivalent other target mechanism in a running communication system [Frö+16].

### 3.2.2 Multi-Mechanisms

Multi-mechanisms are composed of multiple mechanisms according to Figure 3.1. It shows three arbitrary layers with one multi-mechanism per layer and depicts the role of multi-mechanism adaptation. A system configuration consists of coherent mechanisms within multiple multi-mechanisms on multiple system layers. Multi-mechanism adaptation exchanges system configurations to adapt to a given application scenario and to reach the desired quality of service while

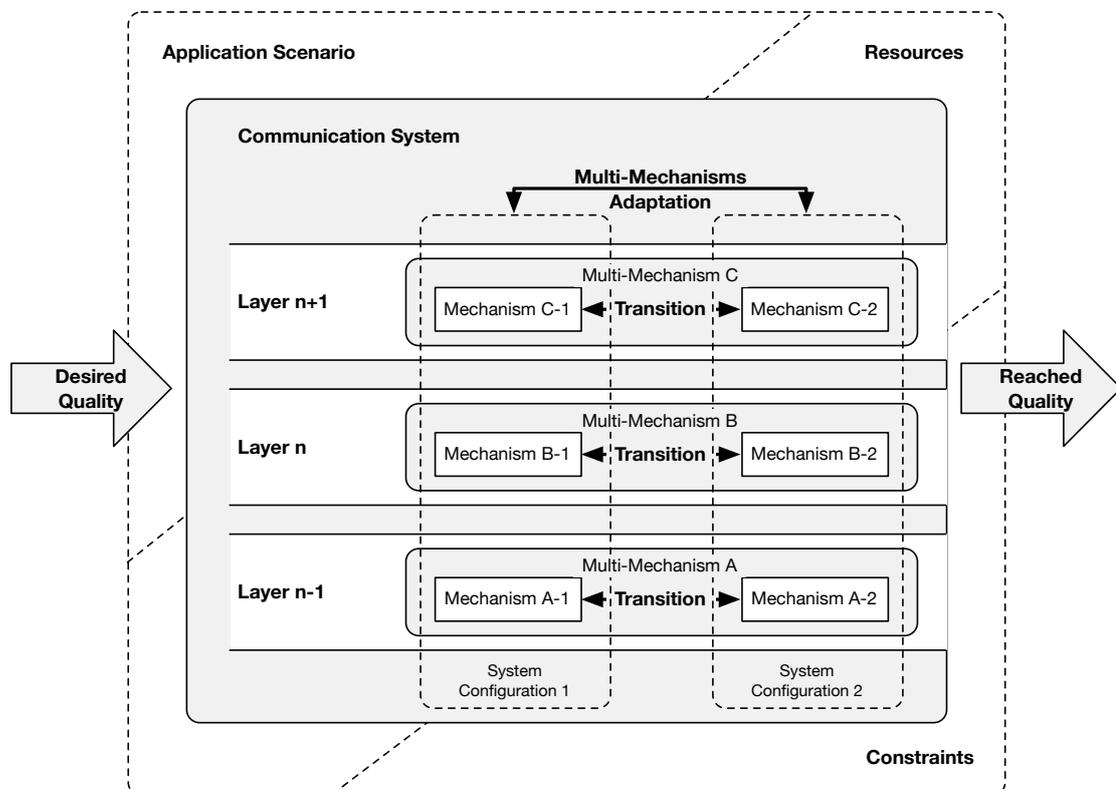
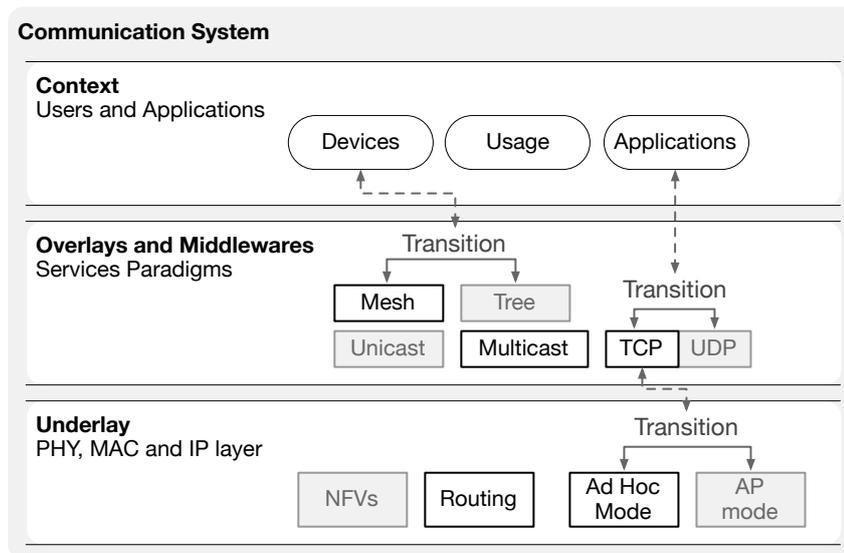


Figure 3.1 Multi-mechanisms in a communication system according to [Frö+16].



**Figure 3.2** Multi-mechanism adaptation across layers in a communication system.

not exceeding the available resources or invalidating other constraints. The depicted layers do not necessarily correspond to the OSI reference model.

**Multi-Mechanism** A multi-mechanism encapsulates several mechanisms and the transitions between them [Frö+16].

The OSI layers originally intended to structure the application layer into layer five (session) and layer six (presentation) are considered obsolete by most researchers today. Instead, in today's Internet, recurrent distributed functionality for applications on top of transport layer has been developed in context of communication paradigms (e.g., remote procedure calls, publish/subscribe systems), overlays (e.g. peer-to-peer), and middleware. In a communication system, a middleware encapsulates recurrent distributed functionality on top of the operating system but below the application environment [Aik+00]. It denotes implementations like messaging systems and distributed task queues that act as a bridge between applications and the network stack of the operating system.

Figure 3.2 depicts an example of multi-mechanism adaptation across different layers of a communication system. The topmost layer describes the applications that make use of the communication system and the network's context. According to Bellavista et al. [Bel+12], a network's context is a four-dimensional space for modeling and describing the environment of a system. It is composed of (i) the computing context that deals with all technical aspects related to computing capabilities and resources, (ii) the physical context that represents the physical reality as it is measured by physical sensors, (iii) the time context that captures the time dimension for sporadic and recurring events, and (iv) the user context that deals with aspects related to the social dimension of the user. All these aspects of the context can be used as input for transition controllers and as the basis for decision making. The middle layer shows overlays and middlewares as bridges between applications and the transport layer of the network stack. The lower layer shows the network underlay, i.e., the physical infrastructure

that delivers packets across networks and that includes the OSI layers network (layer 3), data link (layers 2), and physical (layer 1). In the example shown in Figure 3.2, the usage patterns of the current users in the network are used to trigger an adaptation on multiple network layers. Here, the best way to reach the desired quality is to switch the network overlay to a meshed topology, including the UDP protocol at the transport layer and the Wi-Fi ad hoc mode in the network underlay.

## 3.3 Transitional Computing

Although the terms *mechanism* and *multi-mechanism* are prevalently used for communication purposes, their concept already includes the distributed computation support of interconnected communication systems like overlays and middlewares [Frö+16]. In other words, although its initial purpose is a communication system, the definition of mechanism transitions already includes the functionality of a distributed computing system.

**Distributed Computing System** A distributed computing system is a computing system where its distributed computing and storage resources transparently appear as the resources of a single computer [ST16].

In this definition, the central term is distribution transparency. It is detailed and defined in the Reference Model of Open Distributed Processing (RM-ODP) [Kil+13]. It includes (i) access transparency that hides differences in data representation due to different software and hardware architectures from the user, (ii) location transparency that hides the location of a resource or a process, (iii) concurrency transparency that hides that a resource or a process is accessed by several independent users, (iv) failure transparency that hides failures and recoveries of resources or processes.

Although the term distributed computing system is often used synonymously with the term middleware (e.g., in [ST16]), the definition above can include mechanism transitions at all layers of the computing system as long as the distribution transparency property is not violated. Therefore, the novel idea of transitional computing considers mechanism transitions from the perspective of a distributed computing system. Similar to the concept of a mechanism in communication systems that relaxes the rigid structuring in OSI layers, transitional computing relaxes the rigid structure of distributed computing systems as a middleware between applications and the operating system.

**Transitional Computing** A computing paradigm that extends the scope of mechanism transitions to a distributed computing system by introducing novel transition types (computational transitions) that can be combined with existing transition types (network transitions).

In particular, transitional computing introduces the following three kinds of computational transitions (CT 1-3).

**CT 1 Transitions between Locations:** In a distributed system, services and functions can be applied on different virtual or physical hosts (*locations*). Transitions between locations can improve the quality of a service, e.g., when the response times of a service are reduced, by placing it closer to the user. Several services typically run for a long time without interruption and therefore require mechanisms for performing live migrations. On the other hand, (serverless) functions are typically activated on an input event and terminated after execution, so that downtimes can be used for a relocation to another host.

**CT 2 Transitions between Implementations:** A function, module, or class can have multiple *implementations* with different runtime properties. A transition between implementations is performed to benefit from a specific property of a particular implementation, e.g., a reduced runtime complexity. Here, the term *implementation* means that a programmer has to provide several instances of exchangeable algorithms that solve a specific task differently. However, switching the target architecture of a single implementation is a different kind of transition.

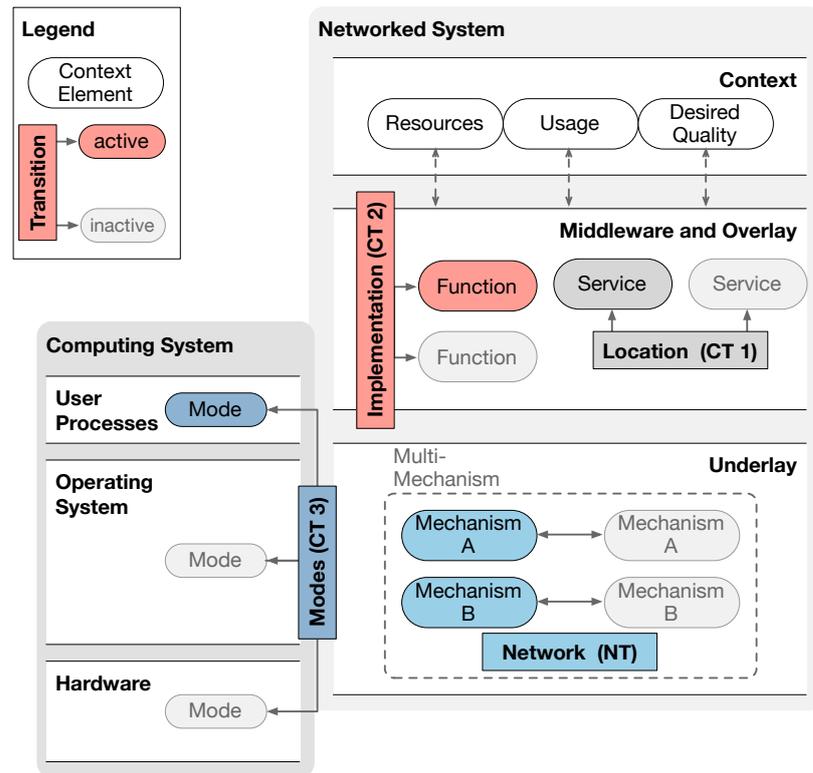
**CT 3 Transitions between Modes:** An implementation of a function or a service can be run in different *modes*, i.e., in different software or hardware layers of a computational system that provide different capabilities and characteristics with respect to functionality, power consumption, and performance. To benefit from the properties of a specific mode, a transition between modes can be performed. Mode transitions are realized by recompiling (interpreting) the implementation of a function or a service for (on) a different target architecture.

Transition types in transitional computing include the novel computational transitions that are described above, as well as network transitions (NT).

**NT Network Transitions:** Network mechanisms primarily provide means for communication, e.g., for exchanging messages, sending and receiving frames, packets or streams of data. Computational and network transitions are deeply connected since services and functions in a distributed computing system need to transfer requests and responses, intermediate results, as well as input- and output data. They can be combined in the form of a multi-mechanism.

Figure 3.3 depicts the concept of transitional computing. The network on the right is structured into three layers: (i) the context of a network, (ii) an overlay and middleware layer, and (iii) the underlay including the OSI network, data link, and physical layer. On the left, the layers of a single computing system like a server or a mobile device are shown: user processes that make use of resources managed by an operating system that in turn uses a binary interface of the underlying hardware.

Transitions between locations (CT 1) are shown in the center of Figure 3.3. They are used to relocate a service or a function to a different physical or virtual host in a network, in order to balance the load of the host systems or to select a host with more appropriate hardware for a service. The relocation includes the transfer of its persistent storage, the content of its memory, and its current state. In cases where large amounts of storage are used, the network needs to provide sufficient bandwidth and a low latency for redirected requests to the service during its



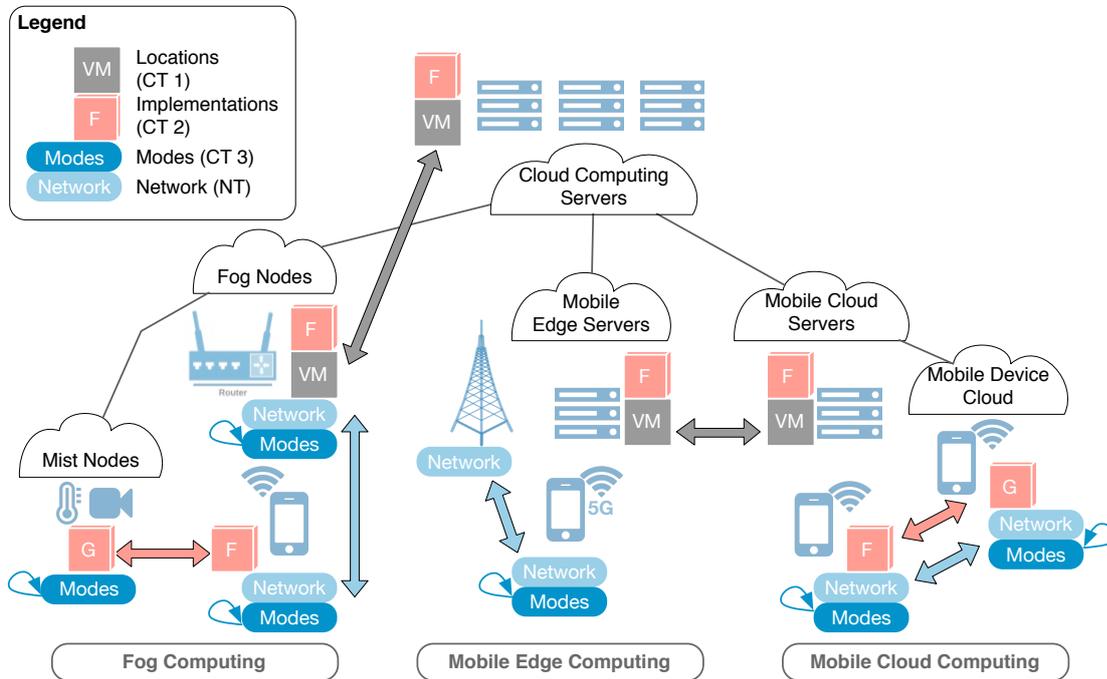
**Figure 3.3** Computational Transitions in a distributed computing system (i) between locations (CT 1), (ii) between implementations (CT 2), (iii) between modes (CT 3), and (iv) in coordination with network transitions (NT).

relocation. Therefore, this kind of computational transitions is typically combined with network transitions in a multi-mechanism.

In Figure 3.3, transitions between implementations (CT 2) reside in the middle layer of a networked system. A function or a service without a strict specification of its realization can have multiple implementations that can be exchanged at runtime. For example, lossy video and audio encodings can be adapted to deliver either a better image quality or a higher compression rate. Applications from the domain of machine learning and artificial neural networks can also make use of different types of algorithms and network sizes. Dynamically exchanging different implementation variants enable the distributed computing system to adapt to the available resources or the required quality of the result at runtime.

Transitions between modes (CT 3) are depicted on the left side of Figure 3.3. Three modes are shown: user mode, operating system mode, and hardware mode. The operating system mode includes OS subsystems such as the process scheduler, device drivers, file systems and the network stack. The hardware mode includes all computing units like co-processors or accelerators within a computing system. The assumption is that the closer the modes are to the bare-metal hardware, the greater are the limitations in their computational capabilities are. Execution environments in these layers can be either a lightweight virtual machine that interprets or just-in-time compiles a virtual instruction set, or a bare-metal environment that executes dynamically loadable modules for its particular platform.

### 3.4 Energy-efficient Transitional Near-\* Computing



**Figure 3.4** Transitions in Transitional Near-\* Computing. The picture shows (virtualized) computing instances (gray), application components at the granularity of services, modules, or functions (orange), network communication paradigms and protocols between hosts (blue), and different modes on the same host (light blue). Arrows indicate where transitions are performed. Computing paradigms related to near-\* computing and their components are labeled accordingly.

Figure 3.4 depicts the big picture of transitional near-\* computing. Despite differences between computing paradigms related to near-\* computing, they share architectural similarities and basic technologies [RLM18]. In Figure 3.4, computing instances like virtual machines (VMs) are provided close to the end user and close to the edge of the network. In these virtualized environments, application-specific services, functions and modules run in bare VMs or on top of domain-specific platforms. Furthermore, resource-constrained devices like mobile and IoT devices are able to perform application-specific tasks on-device in an execution environment without virtualization. They either rely on process-level virtualization or domain-specific programming and execution environments.

Network transitions (NT) are of particular importance at the edge of the network, between access points / base stations and mobile devices, but also in-between mobile devices to address complex and varying network traffic patterns. Mode transitions (CT 3) used to perform near-data, near-storage and near-network computations at the hardware- or operating system layer can reduce the resource consumption of mobile devices. Depending on their hardware capabilities and the kind of data to be processed, battery power, network bandwidth, and CPU processing time can be saved. Location transitions (CT 1) can be used to relocate virtual machines or application-specific tasks to consolidate them on specific (energy-efficient) devices. Furthermore, placing

tasks even closer to the end user, to mobile and IoT devices, reduces latency and saves bandwidth. Adapting the implementation of a service, function, or module (CT 2) adapts a task according to the available resources and hardware capabilities.

Table 3.1 lists the different types of transitions examined in this thesis. They are structured into the three types of computational transitions.

**Table 3.1** Contributions to Energy-efficient Transitional Near-\* Computing.

<b>Transitions between Locations (CT 1)</b>	
Virtual Machine Consolidation	Transitions between virtual machine locations (CT 1) are presented. Consolidating virtual machines on servers in an infrastructure-as-a-service cloud improves its energy efficiency.
Dynamic Role Assignment	Transitions between application- and network service locations (CT 1), combined with network transitions (NT) are presented. In a software-defined wireless network at the network's edge, power consumption can be reduced by adapting the application- and network services accordingly.
<b>Transitions between Implementations (CT 2)</b>	
Multimodal MapReduce	Transitions between data-stream operators (CT 2) in a distributed file system are presented. They can be used to improve the energy efficiency of MapReduce applications that are robust to input variations while preserving the quality of the results.
Opportunistic Named Functions	Transitions between different processing steps (CT 2) during content delivery in a disruption-tolerant network are presented. They are used to trade battery power for network bandwidth and vice versa.
<b>Transitions between Modes (CT 3)</b>	
Multimodal Complex Event Processing (CEP)	Transitions between modes of CEP operators on a mobile device (CT 3) are presented. They reduce the power consumption of CEP operator trees on a mobile device by performing operations near-data on low-power components.
Reactive Programming on Wi-Fi Firmware	Transitions between modes are performed on components developed in a reactive programming language. They can be used to reduce the power consumption in a 802.11 network (CT 3).

## 3.5 Summary

This chapter introduced the novel idea of transitional computing as an extension to mechanism transitions. The novel term of near-\* computing was introduced. It combined four trends focusing on locality of data processing and resource provisioning: near-user, near-network, near-device and near-data. Combined, they form a novel perspective on current computing paradigms summarized as near-\* computing.

Furthermore, transitional computing was defined. In particular, three computational transitions were described:

- Transitions between locations (CT 1)
- Transitions between implementations (CT 2)
- Transitions between modes (CT 3)

Transitions between locations (CT 1) are used to relocate functions and services to other physical or virtual hosts in a network. They are performed to balance the load of the host systems or to select a host with more appropriate hardware for a function/service.

Transitions between implementations (CT 2) refer to the exchange of the implementations at runtime.

Transitions between modes (CT 3) describe transitions between functions/services in different software- or hardware layers (modes) of a computation system. Each mode provides different capabilities and characteristics with respect to functionality, power consumption and performance.

Furthermore, an overview over the contributions to energy-efficient transitional near-\* computing was given.



# 4

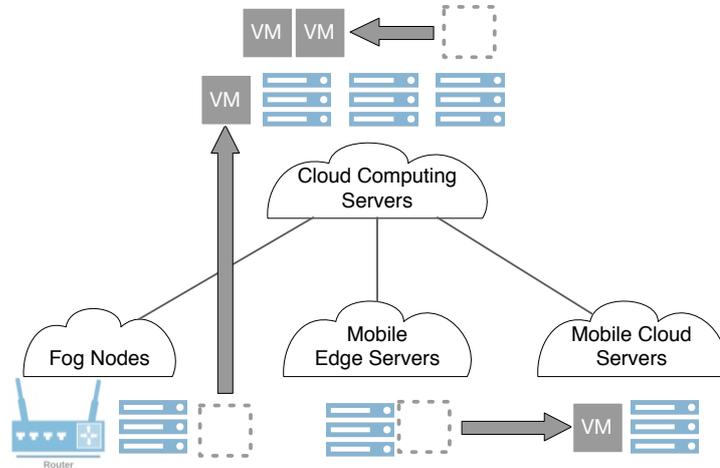
## Transitions between Locations in Near-\* Computing

This chapter presents transitions between locations in near-\* computing. In Section 4.1, two instances of location transitions for near-\* computing are motivated. Furthermore, these instances are presented in detail. Section 4.2 describes the novel approach of virtual machine (VM) consolidation in Infrastructure-as-a-Service (IaaS) clouds. Section 4.3 describes dynamic role assignment in Software-defined Wireless Networks (SDWNs). Finally, Section 4.4 summarizes the chapter.

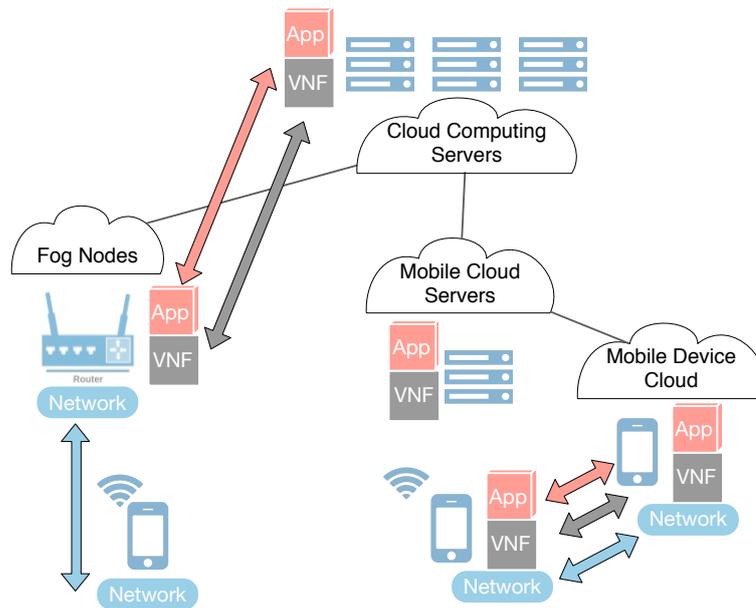
### 4.1 Motivation

In near-\* computing environments, functions, services, or modules of networked applications are executed in proximity to the end user. For this purpose, private cloud infrastructures close to the edge of the network complement dynamical provisioning of computing resources like virtual servers or containers in public cloud infrastructures. Ongoing research for these private cloud infrastructures includes user mobility and demand variation that have been addressed with dynamic service migration and workload scheduling (e.g., [Urg+15; Zha+16]). In contrast to these existing approaches, the novel concept of transitional computing allows the combination of computational transitions and network transitions at different system layers, allowing service migrations and workload scheduling to be complemented with the adaptation of network protocols. Mechanism transitions at several network layers are used to prepare and execute location transitions, further improving the ability of these computing environments to adapt to dynamically varying demands while maximizing the infrastructure's efficiency.

Figure 4.1 and Figure 4.2 depict the virtual machine consolidation and dynamic role assignment approach, respectively, within the context of near-\* computing. Figure 4.1 shows the virtual machine consolidation approach. It addresses private IaaS-clouds as a basis for computing infrastructures in proximity to the end user. Although both intra- and inter-cluster VM migration are supported, it focuses on intra-cluster VM consolidation, allowing more servers within a cluster to power down. Figure 4.2 shows the dynamic role assignment approach. It addresses software-defined wireless networks (IEEE 802.11 WLANs), where application-specific services can be combined with virtual network functions (VNFs) and MAC mechanisms can be tailored to the network topology at IP layer.



**Figure 4.1 Virtual machine consolidation in IaaS-clouds** allows inter- and intra-cluster VM migrations (gray arrows) in near-\* computing based on a shared storage mechanism. The focus is on intra-cluster VM consolidation, allowing more servers to power down.



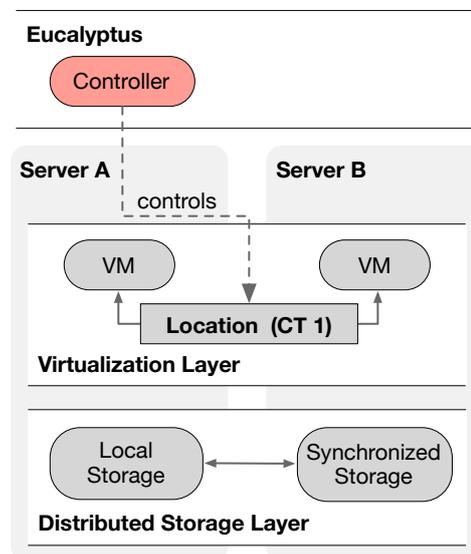
**Figure 4.2 Dynamic role assignment in Software-Defined Wireless Networks (SDWNs)** combines application- (orange) and network services (gray) in near-\* computing with modes on the IEEE 802.11 MAC layer (blue). Roles can be used as building blocks for networked applications.

## 4.2 Virtual Machine Consolidation in IaaS-Clouds

This section introduces a novel approach to virtual machine (VM) consolidation in Infrastructure-as-a-Service (IaaS) environments. First, its relation to transitional computing is described in Section 4.2.1. The following Sections 4.2.2 and 4.2.3 discuss design and implementation issues for an energy-efficient VM consolidation approach in an IaaS cloud. Section 4.2.4 presents experimental results. Section 4.2.5 discusses related work, and Section 4.2.6 concludes this section.

*Parts of this section have been published in [GSF11; GSF13].*

### 4.2.1 Computational Transitions



**Figure 4.2.1** Transitions between locations (CT 1) within virtual machine consolidation in IaaS-clouds.

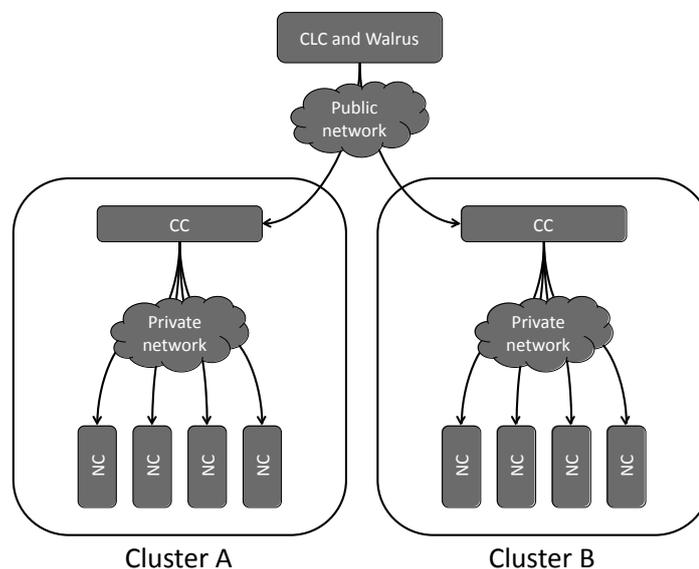
Figure 4.2.1 depicts computational transitions within the virtual machine consolidation approach. At the top, the commonly used cloud computing environment EUCALYPTUS [Nur+09] is shown, an open-source reimplementation of Amazon EC2. Below, a virtualization layer is responsible for managing and providing virtual machines (VMs) on top of physical servers. Transitions between locations (CT 1) based on VM live migrations are initiated by an Eucalyptus controller in order to consolidate the number of active servers and to save power by suspending inactive servers. Live migrations require the virtual disk images on both the source and the destination server to be available. This can either be achieved by using shared storage or by distributing the virtual disk image prior to live migration. In Figure 4.2.1, virtual disk images are distributed prior to live migration via the distributed storage layer at the bottom: To make virtual disks available at the destination server, an on-demand mirroring technique based on distributed replication block devices (DRBD) is used. During normal operation, all read and write operations on the virtual disk image are performed locally (local storage). Only when a live migration is initiated, are block devices mirrored to another host (synchronized storage).

## 4.2.2 Design

EUCALYPTUS is an open-source Cloud Computing system emulating the interfaces of Amazon Elastic Compute Cloud (Amazon EC2)<sup>1</sup>, that provides low level control of computing resources, and Amazon Simple Storage Service (Amazon S3), that provides an object storage. The aim of EUCALYPTUS is to provide an open-source architecture for both public and private Infrastructure-as-a-Service (IaaS) cloud computing platforms. The main design goals of EUCALYPTUS are non-intrusiveness and modularity while achieving the greatest degree of scalability possible. The only requirement for deploying and running EUCALYPTUS, apart from providing one of the supported virtualization environment such as Xen, is the ability to run Web Services. Assuming a working virtualization environment, there are no significant modifications necessary on the underlying system, all EUCALYPTUS components have been conceived and designed as a collection of Web Services. It supports software packages of industry-standard Web Services such as Apache<sup>2</sup>, Axis<sup>3</sup> and Rampart<sup>4</sup> as well as a well-defined interface, defined in the Web Service Description Language. For networking, standard VLAN and DHCP server packages and the Linux operating system interface can be used.

### 4.2.2.1 EUCALYPTUS Architecture

In Figure 4.2.2, the EUCALYPTUS architecture is shown. It reflects the hierarchical nature of computing resources typically available to providers. Usually, system administrators deploy clusters as pools of worker machines on private, unroutable networks with a single head node responsible for traffic between the worker machines and the public network. Therefore, the



**Figure 4.2.2** Overview of the EUCALYPTUS architecture [Nur+09].

<sup>1</sup><http://aws.amazon.com/ec2>

<sup>2</sup><http://httpd.apache.org>

<sup>3</sup><http://ws.apache.org/axis2>

<sup>4</sup><http://ws.apache.org/rampart/c>

design of EUCALYPTUS consists of three hierarchical levels: Node Controller, Cluster Controller, and Cloud Controller [Nur+08]. Furthermore, EUCALYPTUS also includes a Storage Controller (Walrus), an Amazon S3 interface emulation.

In the following, the controller hierarchy is described in detail:

**Node Controller (NC):** A component that is executed on the physical resources that host VM instances and is responsible for instance start up, inspection, shutdown, and cleanup. It is structured as a set of Web Service operations as external interface, a monitoring thread maintaining the state of the hypervisor and its virtual machines as well as resource management routines to provide instance startup and cleanup.

**Cluster Controller (CC):** A component responsible for collecting state information from its collection of Node Controllers, scheduling incoming instance execution requests to individual Node Controllers as well as managing the network.

**Cloud Controller (CLC):** An external user interface and entry-point for administrators. It is the main decision-making component in EUCALYPTUS, designed as a collection of services connected to an Enterprise Service Bus (ESB) [Bau+10].

**Storage Controller (Walrus):** A simple storage service that implements the Amazon S3 interface. It acts as storage service for VM images and can be used to stream data from inside out of the cloud as well as from outside into the cloud.

The scheduling algorithm implemented in EUCALYPTUS is already able to suspend or shutdown idle servers in a cluster. Nevertheless, depending on the usage profile of the cloud, VM relocations are able to increase the number of suspended servers by consolidating VMs on a minimum set of servers. There are several practical problems to solve:

1. **Demand Variations:** Due to the on-demand nature of cloud computing, there is an unpredictable user behavior in terms of starting and terminating VMs. Since such demand changes can even occur during VM migration or its pre- and postprocessing phases, the decision to migrate VMs may prove as wrong and may lead to additional energy consumption.
2. **Live Migration Overhead:** There is an additional overhead spent for live migration itself and for its pre- and post processing phases.
3. **Resource Sharing between VMs:** Consolidated VMs share resources (hardware, caches etc.), i.e., there may be interferences between VMs running on the same server.

These problems may influence the energy efficiency of a cloud significantly. To face them, different approaches can be taken: *either* optimizing the scheduling algorithm by considering the different workloads of VMs and by considering their execution times, *or* by optimizing the software architecture for energy efficiency. The first approach requires additional information originating from performance trackers and may lead to additional overhead, which is often unacceptable for cloud computing providers [SKZ08]. The second approach is the approach proposed here: developing a solution that reduces energy consumption and integrating it smoothly into EUCALYPTUS with a minimum additional performance or energy overhead.

**Table 4.2.1** ACPI Power Consumption

Time spent for a sleep/wake-up cycle and the power consumption in different ACPI states.

State	Time	Power
G0-S0 (100% utilization)	/	110 W
G0-S0 (idle)	/	62 W
G1-S1	7.1s	57 W
G1-S3 suspend-to-RAM	11.2s	6 W
G1-S4 suspend-to-disk (without Xen)	28.8s	4 W
G2-S5	51.8s	4 W
G2-S5 (without Xen)	37.7s	4 W

#### 4.2.2.2 Power Management

In general, power management requires an abstract interface that is independent of hardware vendor specifications. This can be found in the Advanced Configuration and Power Interface (ACPI)<sup>5</sup>, the prevailing power management interface today.

The system states available for ACPI differ in the sleep/wake-up time and in their power consumption. Table 4.2.1 compares both factors for an Intel Pentium CPU. It shows a high difference of 56 W between the states G1-S0 (idle) and G1-S3 (suspend-to-RAM). Setting servers to *suspend-to-RAM* when they are idle ensures both a small sleep/wake-up penalty as well as a small power consumption. This is the basis for the energy savings proposed here.

#### 4.2.2.3 Storage Synchronization

In contrast to network file system approaches, such as accessing disk images using the Network File System (NFS) protocol, EUCALYPTUS does not need permanent remote access to a file server via a network. In fact, this would lead to network traffic during the access to a disk image. Even if there were no ongoing live migrations, there would be an overhead.

To avoid shared storage, Distributed Replicated Block Devices (DRBD)<sup>6</sup> are used. This concept originated from high-availability computing and can be used for data storage in a distributed system. DRBD replicates data storage to different locations in a stable, fault tolerant way. The DRBD module can operate in two modes: stand-alone and synchronized. In stand-alone mode, all disk accesses are simply passed to the underlying disk driver. In synchronized mode, disk writes are both passed to the underlying disk driver and sent to the backup machine via a TCP connection. Disk reads are served locally.

Since Armbrust et al. [Arm+10] have identified data transfer bottlenecks as an obstacle for cloud computing services, network traffic through storage synchronization should be reduced. Therefore, the synchronization of gigabyte-sized disk images is based on the multi-layered root file system (MLRFS) approach developed by Schwarzkopf et al. [Sch+09]. They used a MLRFS to centrally apply security updates and to reduce disk image transmission time. Therefore, only small, separate layers instead of the whole, potentially large disk images are transmitted.

<sup>5</sup><http://www.acpi.info>

<sup>6</sup><http://www.drbd.org>

All layers are overlaid transparently and form a single coherent filesystem using a Copy-On-Write (COW) mechanism. In our approach, a read-only base layer containing the disk image downloaded from Walrus is used in combination with a writable memory disk, covering read-/write-accesses to the root file system. Live migration provided by Xen automatically synchronizes the virtual memory between source and destination host, i.e., there is no need for additional replication here.

The size of the memory disk restricts VM reconfigurations as well as additional software installations. On the other hand, most of the software needed at runtime is installed *before* it is uploaded to Walrus, hence the installations made in a running VM are lost after its termination due to the EUCALYPTUS instance cleanup process.

As described by Haselhorst et al. [Has+11], both mechanisms can be used for live migrations in cloud computing environments. From an energy perspective, both are very efficient, as indicated by our experiments. Stand-alone DRBD devices and a MLRFS do not increase the power consumption of a server. Until the relocation of a VM is initiated, the power consumption of a server is not affected.

#### 4.2.2.4 EUCALYPTUS Integration

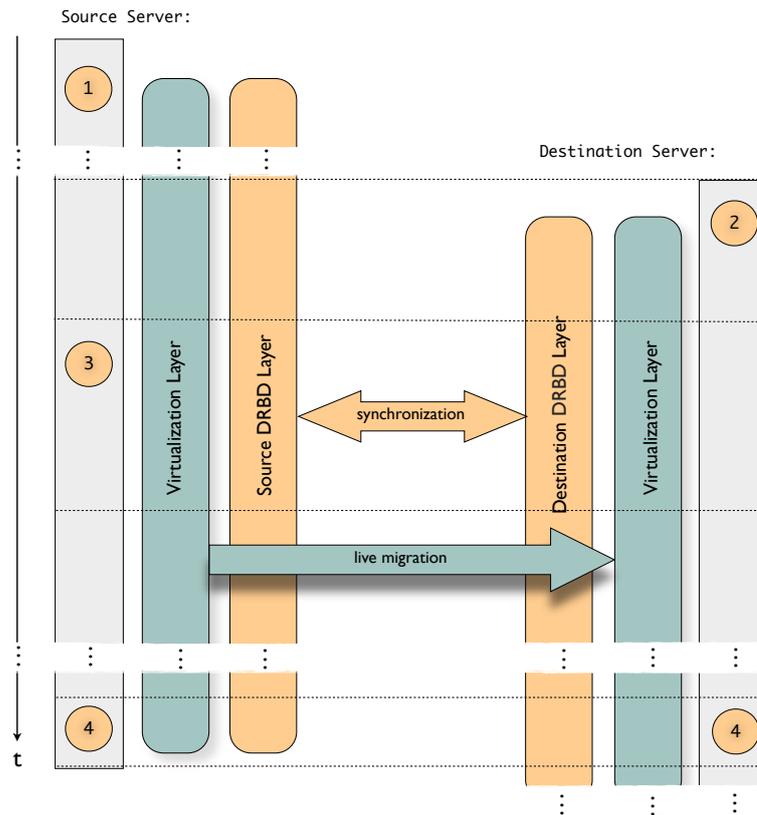
A EUCALYPTUS instance consists of a kernel/ramdisk tuple, a root file system, swap space, potentially an ephemeral device as well as network interfaces. In the following, the term *instance* is used for EUCALYPTUS instance. Analogically, the term *instance relocation* is used for the relocation of a whole instance, including VM live migration and the transformation of its state.

Instance relocations are initiated and controlled by the Cluster Controller. For this purpose, two components are added: A relocation agent and an instance relocation algorithm.

The relocation agent is a separate component inside the Cluster Controller. It is responsible for providing an interface enabling other components to initiate and abort instance relocations, handling state transitions and events during the instance relocation process. However, the relocation agent is strongly connected to the monitoring thread, which is modified to periodically poll the state of ongoing instance relocations. There are two major design goals for the relocation agent: It should be reliable, i.e., timeouts of all kinds, Node Controller errors or the loss of a connection should not lead to an undefined state. Furthermore, it should be scalable. The overhead should grow linearly with the number of relocations.

The instance relocation algorithm follows a centralized server approach, reflecting the architecture of EUCALYPTUS. Parameters such as the interval the instance relocation algorithm runs and, obviously, the type of the instance relocation algorithm should be configurable by the EUCALYPTUS administrator. In general, the instance relocation algorithm has to supervise the available set of servers, their state as well as their set of available resources such as the number of virtual CPUs, the amount of virtual memory and disk space.

Based on this information, the algorithm initiates instance relocations that lead to additional server suspensions. It takes into account general and EUCALYPTUS-specific restrictions for instance relocations: in general, only running instances are relocated. VM instances that have been prepared and not started yet as well as instances currently terminating are ignored.



**Figure 4.2.3** Temporal course of a VM relocation process.

Figure 4.2.3 shows the interconnection between source and destination server during the instance relocation process. Each step is initiated by the Cluster Controller relocation agent, while the Node Controller supervises the operations on the corresponding server:

1. **Instance Startup:** The disk, kernel and ramdisk images are downloaded from Walrus. MLRFS support is added and the underlying DRBD devices are initialized, before the instance is started.
2. **Pre-Processing:** On the destination server, the preparation of the instance relocation process is similar to an instance startup process: The images are downloaded from Walrus, MLRFS and DRBD devices are configured.
3. **Migration:** The DRBD devices are synchronized, afterwards the live migration from source to destination server is performed.
4. **Post-Processing:** On the source server, all remaining data is removed. On the destination server, the instance is activated and maintained by the destination Node Controller.

### 4.2.3 Implementation

The implementation of the proposed approach is based on the EUCALYPTUS version 1.6.2 and the Xen version 3.2.1<sup>7</sup>. In particular, the Node Controller implementation is integrated into the Xen-specific parts of the EUCALYPTUS Node Controller.

#### 4.2.3.1 Cluster Controller

In general, the Cluster Controller is written in the C programming language, using Axis2/C Web Service operations to communicate with the Node Controllers. Instance relocations are transparent to the Cloud Controller; it is not affected in this implementation.

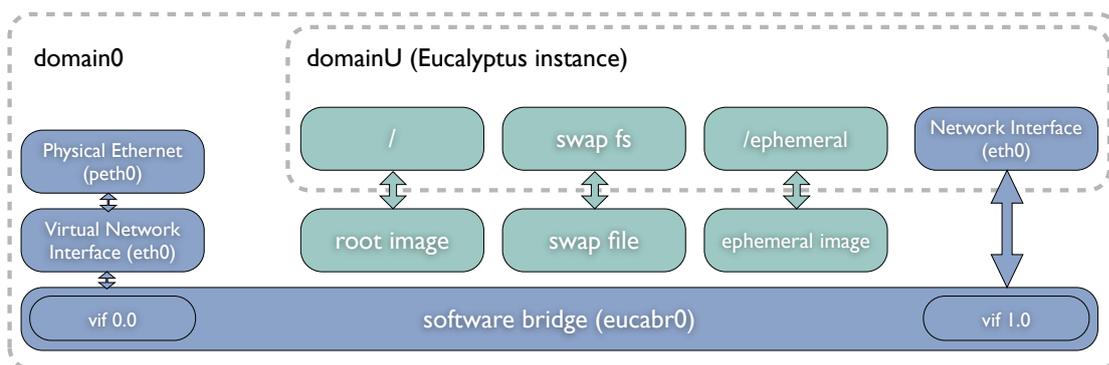
The instance relocation algorithm uses a resource map, basically consisting of a mapping between servers and instances after all relocations have been done.

The relocation agent handles state transitions of a relocation. It consists of two functions to initiate and abort instance relocations and a state machine for handling state transitions and events. The state is stored in a separate cache and is periodically updated by the monitoring thread, which polls the state of ongoing instance relocations from the Node Controller by using a newly integrated DescribeRelocation operation. The relocation agent periodically checks its cache and initiates the pre-processing, migration and post-processing phases. Multiple instance relocations can be handled concurrently.

#### 4.2.3.2 Node Controller

Figure 4.2.4 depicts the images and network interfaces arranged by the EUCALYPTUS instance startup process. The Xen bridged networking configuration in combination with the Linux build-in support for ethernet software bridges<sup>8</sup> is used.

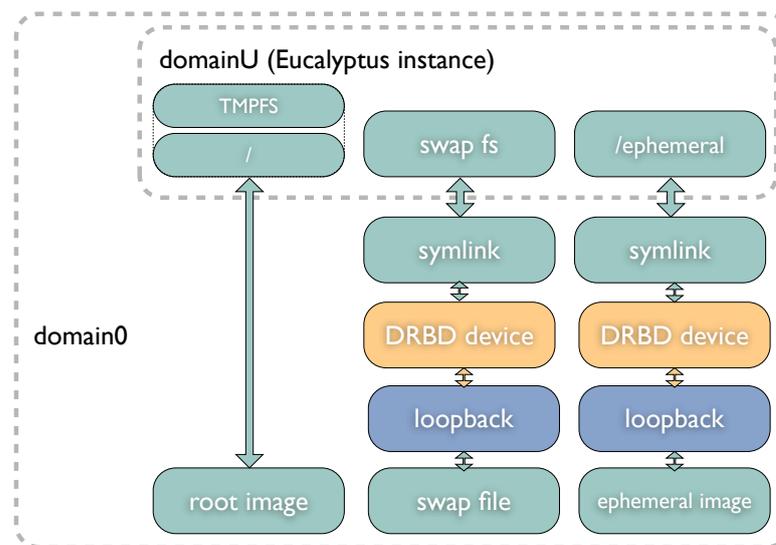
The synchronization between Primary and Secondary DRBD devices is achieved by the synchronous replication (DRDB protocol C) configuration. The network bandwidth used during live migration as well as the TCP ports connecting the DRBD devices are configurable.



**Figure 4.2.4** Scheme of the network interfaces and images used for an EUCALYPTUS instance, running on a Xen hypervisor.

<sup>7</sup>Xen supports host-based suspend-to-ram since version 3.2; it is required for this implementation.

<sup>8</sup>In Linux, the `brctl` command connects multiple networks and network segments by forwarding via MAC-address.



**Figure 4.2.5** Scheme of a modified EUCALYPTUS instance.

In Figure 4.2.5, the modified EUCALYPTUS instance is shown. At the bottom, similar to the previous Figure 4.2.4, the VM root images, the swap file, and the ephemeral image are shown. The readonly root image is overlaid by a temporary file system (TMPFS) with a configurable size using an AUFS file system, covering read-/write-accesses to the root file system. While the root image is distributed via the Walrus Controller, the swap file and ephemeral image need to be synchronized via DRBD devices. The Xen migration process requires identical device names (e.g., loopback mounted disk images) on both source and destination server. Since all resources, including loopback devices and DRBD devices, are acquired independently by the Cluster Controller and other Node Controllers, their names can overlap. Therefore, symbolic links pointing to the device file system are used to cover the real device names. This ensures normal access to the real device as well as arbitrary naming of the symbolic link file. The symbolic links are stored in the user instance directory and are further used for Xen configuration.

In the following, the different phases of the instance relocation process according to Figure 4.2.3 are described.

### Instance Startup

The `RunInstance` operation is modified in order to modify both the Xen configuration for multi-layer root file system support and the swap and ephemeral disk creation for initializing the underlying DRBD devices. For this purpose, loopback devices are initialized, DRBD metadata is created and the DRBD devices are attached to the loopback devices. Since DRBD devices typically use a meta-disk to store metadata information, the DRBD meta-disk in this implementation consists of a 128 MB sparse image that is stored in the meta-disk's local user instance directory. During the VM startup process, a separate Perl helper script `gen_libvirt_xml` is used to generate the VM configuration file.

### Pre-Processing

A new `RelocationPreparation` operation is integrated into the Node Controller Web Service, which is responsible for preparing the disk images and the Xen configuration on the destination

host. As a parameter, the instance state (containing the VM configuration, e.g., the disk image location in Walrus or the network configuration) is transferred. The root file system is either downloaded via Walrus or copied from the local instance cache, the Xen configuration is created, and the network and the DRBD devices are initialized. This process is performed asynchronously, its state is polled by the Cluster Controller. The thread waits for the end of the device synchronization. After that, all DRBD devices are set to dual-primary mode.

### Migration

The newly integrated `RelocationMigration` operation signals to the source Node Controller that the destination Node Controller is ready for migration. The TCP ports are used to set up the connection between Primary and Secondary DRBD devices, and the synchronization is started. The thread waits for the end of the device synchronization and starts the live migration using the `libvirt virDomainMigrateToURI()` function afterwards. Since supervising the state of an instance during its migration may lead to failures, state monitoring is stopped during the live migration. Therefore, the `Node Controller monitoring_thread` is modified.

### Post-Processing

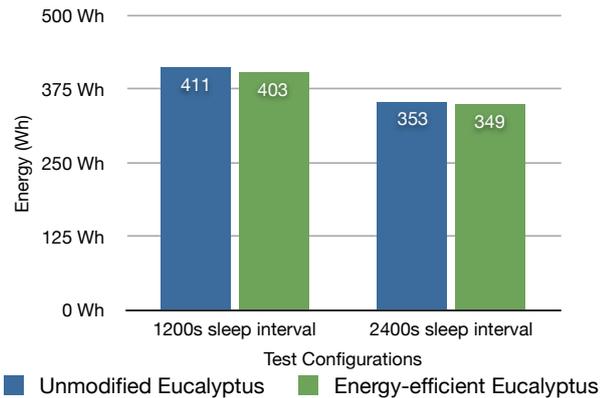
The newly integrated `RelocationCommit` operation signals to both source and destination Node Controller that the live migration process has been completed. On the source Node Controller the relocated instance has to be removed, on the destination Node Controller the relocated instance has to be added to the instance cache. Resources allocated during the relocation process are freed and the remaining DRBD devices are disconnected. The source Node Controller's cleanup process, i.e., the removal of the local instance directory, the DRBD device detaching from the loopback devices and their metadata removal are performed by the `monitoring_thread`.

#### 4.2.4 Experimental Evaluation

To demonstrate that the proposed approach indeed saves energy, a set of measurements were conducted. This includes several short- and long-term tests based on instance workloads produced with common operating system benchmarks, web-server emulations, and different MapReduce applications. In the following, live migration is limited to clusters rather than a set of clusters in a cloud. The hardware configuration consists of Intel Pentium CPUs test servers equipped with 4096 MB memory. The startup/terminate experiments are performed on two servers, the other experiments with three servers. The power and energy measurements are taken with an EasyMeter<sup>9</sup> electricity meter. It is connected to a separate ethernet network via the Multi Utility Communication (MUC) tool and polled at a 5 Hz rate. All EUCALYPTUS images are Xen compatible Debian Linux images, with an instance configuration of one virtual CPU, 384 MB memory and 3 GB total disk space, with a root file system size of 1537 MB and an ephemeral device size of 1019 MB. Additionally, 512 MB swap space is available. An enabled multi-layer root file system works with a 196 MB sized writeable layer and a DRBD device with a synchronization rate of 50 MByte/s.

---

<sup>9</sup><http://www.easymeter.com>



**Figure 4.2.6** Energy consumption of the Startup/Terminate experiment for two test configurations.

#### 4.2.4.1 Startup/Terminate Experiments

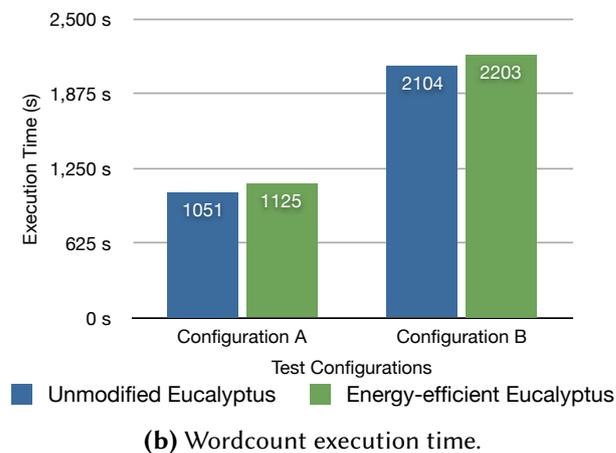
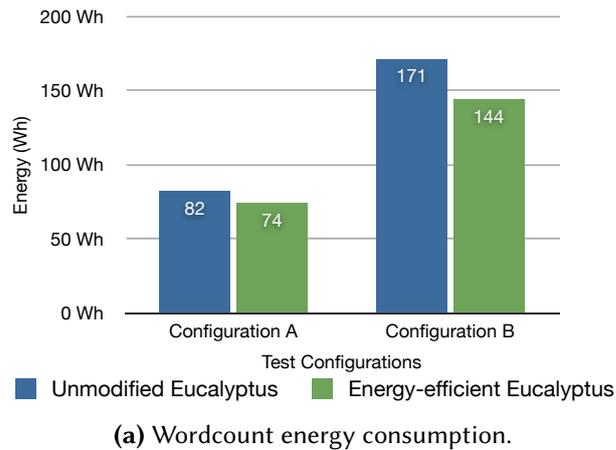
The first experiment is intended to observe the power and energy consumption of a EUCALYPTUS cloud with randomly generated user requests. Its basic principle is to send requests from an external client to the EUCALYPTUS cloud, either to start or terminate a random number of instances alternately. The only workload created by the instances are CPU, memory and disk operations during boot time. Two different measurement configurations were used: One with a random sleep interval of 1200 seconds between client operations, the other one with a 2400 seconds sleep interval. The sleep interval ensures that there is enough time between a startup/terminate sequence to actually observe power and energy consumption.

The experimental results shown in Figure 4.2.6 indicate that the instance relocation algorithm does not lead to an increased energy consumption. The reduction between one and two percent of the energy consumption for the energy-efficient EUCALYPTUS in comparison to an unmodified EUCALYPTUS is due to small differences in power consumption of the different servers (between one and two Watts). The energy-efficient EUCALYPTUS coincidentally relocated instances to a server with the smaller power consumption.

The differences in energy consumption between the two configurations stem from the different number of instances started during the experiment (21 and 13). First, power consumption peaks can be observed in correlation to the VM boot process during the start of an instance. Second, the total number of instances running in the first configuration is often higher than in the second configuration. In the latter, the servers can be suspended more often, which leads to a smaller energy consumption.

#### 4.2.4.2 MapReduce Experiment

This experiment monitors energy consumption of an EUCALYPTUS cloud during the execution of a distributed application. Since MapReduce is widely used in cloud computing environments (such as Amazon Elastic MapReduce), it is a good candidate for this type of experiment. In the following, an Apache Hadoop MapReduce implementation in combination with a Hadoop



**Figure 4.2.7** MapReduce Wordcount Experimental Results.

File System (HDFS)<sup>10</sup> is used. As a distributed file system, HDFS spreads data on different nodes to increase data availability and reliability. The typical configuration of a MapReduce consists of one master node, controlling several slave nodes. The master serves as NameNode and JobTracker, while the slaves are serving as DataNodes and TaskTracker.

The test applications are automatically installed and initiated by an external client on previously started EUCALYPTUS instances inside the test cloud. The maximum number of tasks spawned simultaneously on a TaskTracker are set to its default, two mappers and two reducers on a TaskTracker. All test data, the HDFS as well as the Hadoop installation itself are stored on the ephemeral device of the EUCALYPTUS instance. Initially, the instances are placed on three different servers: *Server 1* hosts two instances (one JobTracker and one TaskTracker), *Server 2* hosts two instances (one TaskTracker and an idle instance) and *Server 3* hosts one instance (one TaskTracker). After the experiment has been started, the idle instance on *Server 2* is terminated.

<sup>10</sup><http://hadoop.apache.org>

## Wordcount

The Wordcount experiment is aimed to show the processing of a large amount of data in a cloud. The Wordcount application reads a text file and counts how often a word occurs. The test data consist of the complete Linux kernel 2.6.33.1 including source code, configuration files and documentation, concatenated and stored in a single file. It is performed with two different test configurations:

- A. Wordcount is performed on text files with 110.4 MB total size 10 times in a row.
- B. Wordcount is performed on text files with 110.4 MB total size 20 times in a row.

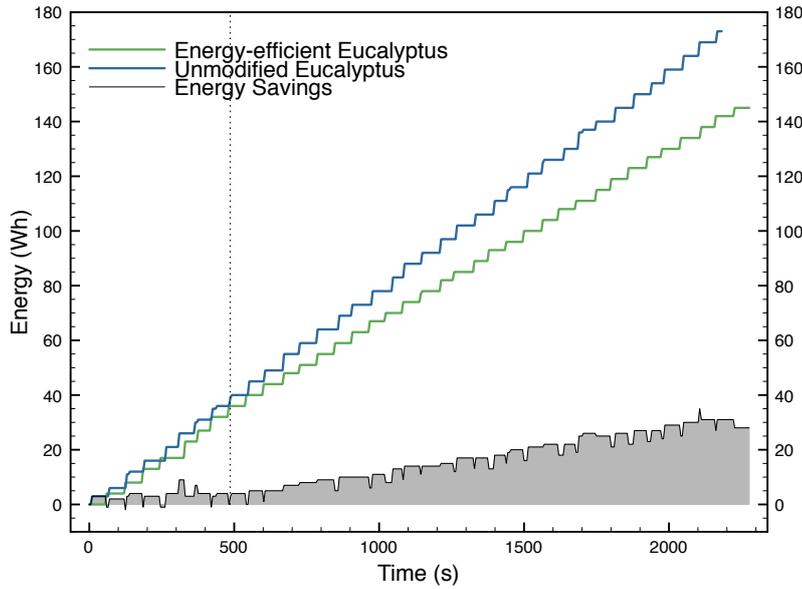
Figure 4.2.7a shows the energy consumption of Wordcount in both test configurations. The energy-efficient EUCALYPTUS finishes 80 seconds after *Server 3* is suspended, resulting in a reduced energy consumption of 8 Wh, i.e., 10% of the total energy consumption of the unmodified EUCALYPTUS. In *configuration B*, our implementation saves 27 Wh, i.e., 16% of the energy consumed by unmodified EUCALYPTUS.

Figure 4.2.7b shows the execution time of Wordcount in both test configurations. For both configurations, the execution time of the energy-efficient EUCALYPTUS is increased (*configuration A*: 7%, *configuration B*: 5%). The important part of the difference between the unmodified EUCALYPTUS and the energy-efficient EUCALYPTUS comes from the instance relocation process: the parts of the test executed during DRBD synchronization and VM live migration are up to 30% slower than the other parts. In absolute values, the execution time is increased by about more than 40 seconds. In fact, a 30% slower execution time may not be acceptable in some cases. But in long-term calculations MapReduce is typically used for, this does not dominate the total execution time.

The absolute execution time of Wordcount is increased between both *configuration A* and *configuration B*. The parts of Wordcount executed after the instance relocation are slightly increased (smaller than one percent compared to the unmodified EUCALYPTUS). This can be interpreted as a slight performance degradation due to resource sharing on *Server 2*. In fact, in the long term, the performance degradation is about 5%.

Figure 4.2.8 shows the energy consumption over time for both the unmodified EUCALYPTUS and the energy-efficient EUCALYPTUS in *configuration B*. Since both configurations show a similar pattern for both energy and power consumption, *configuration A* is not depicted. The filled region in the figure shows a difference between both energy consumptions. This can be interpreted as a potential energy saving for time  $t$  if the test was finished at  $t$ .

There are two turning points in the figure: The first one is marked with a vertical dotted line and describes the moment when *Server 3* is suspended. Up to this moment, the potential energy saving is only fluctuating (due to a varying actualization rate of the electricity meter). Afterwards, the gradient of the energy-efficient EUCALYPTUS decreases while the gradient of the energy consumption of the unmodified EUCALYPTUS remains the same. The second turning point is the moment when the unmodified EUCALYPTUS test is completed. Until then, the difference between the energy-efficient EUCALYPTUS and the unmodified EUCALYPTUS is 35 Wh. Afterwards, the remaining execution time of the energy-efficient EUCALYPTUS reduces the potential energy savings. Finally, the energy saved amounts to 27 Wh.



**Figure 4.2.8** Wordcount energy consumption trend for *test configuration B*.

### Estimation of $\pi$

The PiEstimation experiment is aimed at evaluating CPU and memory-intensive operations in a cloud. Therefore, a MapReduce program to estimate the value of Pi using the quasi-Monte Carlo method is performed with two different test configurations:

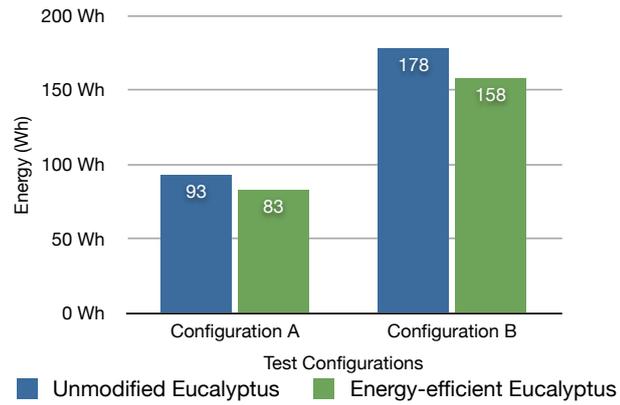
- A. PiEstimation is performed with 1,000 maps and 1,000,000 points.
- B. PiEstimation is performed with 2,000 maps and 1,000,000 points.

Figure 4.2.9a shows the energy consumption of PiEstimation in both test configurations. Although *configuration A* finishes 60 seconds after *Server 3* is suspended, our implementation saves 8 Wh, i.e., 10% of the total energy consumption. Furthermore, our implementation saves 27 Wh in *configuration B*, which is 16% of the total energy consumption.

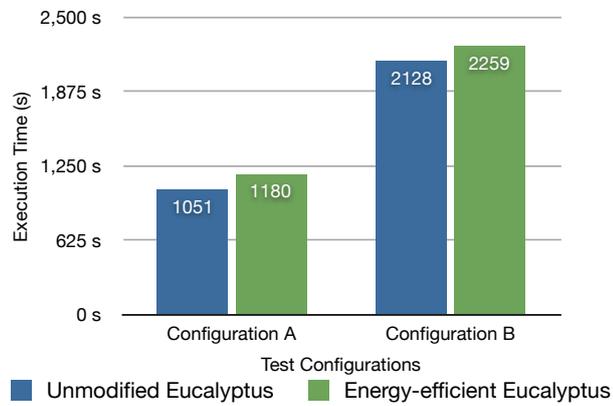
Figure 4.2.9b shows the execution time of PiEstimation in both test configurations. The execution time of the energy-efficient EUCALYPTUS is increased in both *configuration A* (about 12%) and *configuration B* (about 6%). The difference between the execution times of the unmodified EUCALYPTUS and the energy-efficient EUCALYPTUS is about 130 seconds, for both configurations.

Figure 4.2.9c shows the energy consumption over time for both the unmodified EUCALYPTUS and the energy-efficient EUCALYPTUS *configuration B*. Similar to the Wordcount results, the gradient of the energy consumption of the unmodified EUCALYPTUS does not change, but after the suspension of *Server 3*, the gradient of energy-efficient EUCALYPTUS decreases. The suspension of *Server 3* leads to a growing energy saving potential of up to 32 Wh. After the turning point for the energy saving potential, the potential energy savings are reduced. Since the power

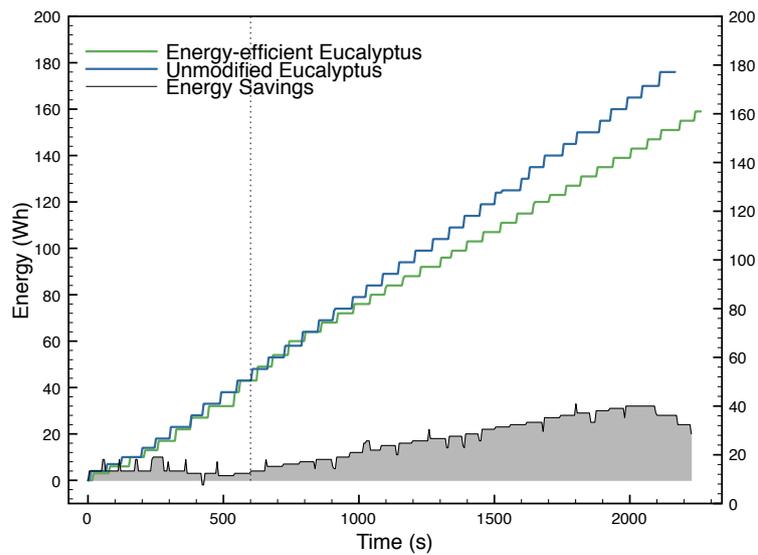
## 4 Transitions between Locations in Near-\* Computing



(a) PiEstimation energy consumption.



(b) PiEstimation execution time.



(c) PiEstimation energy consumption trend for *test configuration B*.

**Figure 4.2.9** MapReduce PiEstimation Experimental Results.

consumption difference between G1-S0 (idle) and G1-S3 (suspend-to-RAM) is very high (about 52 W), the additional energy spent for longer execution times is amortized very fast, even with more unfavorable configurations.

The average power consumption of the unmodified EUCALYPTUS was 286.5 W, while the average power consumption of the energy-efficient EUCALYPTUS was 250.3 W. Furthermore, after the suspension of *Server 3*, the average power consumption of the energy-efficient EUCALYPTUS was reduced to 235.3 W. This means that the average power consumption of each server was 117.6 W, compared to 95.5 W of the unmodified EUCALYPTUS. In other words, the remaining servers run more energy-efficiently; they are highly utilized and their power consumption comes close to their maximum. Due to DRBD device synchronization, there is a power peak of about 353 W, lasting for 136 seconds.

#### 4.2.4.3 KernCompile Experiment

Linux kernel compiles are the basis for commonly used CPU throughput benchmarks like *kernbench*<sup>11</sup>, designed to compare different operating system kernels on the same machine or different hardware. Kernel compiles are often used as benchmarks because their workload combines CPU-, memory- as well as I/O-intensive parts. For example, kernel compiles are used to show the slowdown of a live migration [Cla+05]. In our test, a vanilla 2.6.33.1 kernel was compiled with the default configuration and four jobs compiling concurrently. The results of this experiment are: At first, the energy consumed for an instance relocation is very small. In this disadvantageous experimental setup, the additional energy consumed is about 3 Wh, which is only two percent of the total energy consumption. Second, the experimental results confirm the conclusions of the Startup/Terminate experiment. Even though there are power consumption peaks due to DRBD device synchronizations and VM live migrations, instance relocation itself is not energy-intensive.

#### 4.2.4.4 Apache Benchmark Experiment

The Apache Benchmark<sup>12</sup> is a simple HTTP server benchmark typically used to show how many requests per second the Apache installation is capable of serving. For this purpose, a client repeatedly fetches static web pages from a web server. This experiment is aimed at emulating a web server application typically used by a cloud customer. Due to the concurrent requests handled by the Apache application, there is a lot of interprocess communication. In general, this test is more operating system-intensive rather than I/O-intensive.

The results of this experiment are similar to the KernCompile tests described in the previous section. The difference in the average power consumption between this experiment and KernCompile is only about 3 W in all test configurations, apart from the reduced power consumption of 3 W and fluctuations in the electricity meter measurements. In addition, the pattern of power consumption in the different phases of the experiment is similar to the KernCompile power consumption.

---

<sup>11</sup><http://ck.kolivas.org/apps/kernbench>

<sup>12</sup><https://httpd.apache.org/docs/2.4/programs/ab.html>

### 4.2.5 Related Work

Server consolidation is an approach to maximize resource utilization while reducing energy consumption. There are several studies about different aspects of server consolidation in a cloud computing environment.

Srikantaiah et al. [SKZ08] studied the inter-relationships between energy consumption, resource utilization, and performance of consolidated workloads. The consolidation problem is modeled as a modified bin packing problem. However, this approach is application and workload dependent. In contrast, our approach is based on VMs, allowing to control the entire software stack from the kernel upwards, leading to generic workloads. This is what is usually offered by a cloud computing service such as Amazon EC2.

Verma et al. [VAN08] examined a power and migration cost-aware application placement controller in virtualized heterogeneous systems, minimizing power subject to a fixed performance requirement. The authors designed and implemented *pMapper*, a power-aware application placement framework that continuously optimizes the allocation. VMs are live migrated from one server to another, the performance degradation due to migrations is taken into account. However, no details about the method for calculating the migration costs are presented. In contrast to our work, the mechanisms for live migration itself are not evaluated.

Energy efficiency was addressed by Nathuji et al. [NS07] on the hypervisor level. *VirtualPower*, a Xen hypervisor implementation to control and coordinate the effects of VM-level power management policies on virtualized resources has been developed. It provides soft and hard techniques, referring to software solutions like limiting the hardware usage of a VM or scaling the frequency of a processor, respectively. To achieve this, Nathuji et al. implemented artificial *VirtualPower* Management (VPM) states, used by VPM mechanisms and serving as a base for more complex VPM rules. Their approach relies on evaluating a global consolidation strategy based on VM live migration and VPM rules, aimed at mapping multiple instances of virtual resources to appropriate efficient physical resources. However, the durations of VM live migrations used in this work are quite short due to the migration of local persistent storage. In contrast, EUCLYPTUS or Amazon EC2 offer gigabyte-sized disk images to the user.

Van et al. [VTM10] proposed a resource management framework combining a utility-based dynamic VM provisioning manager and a dynamic VM placement manager. The paper describes VM live migration as a technique used for maximizing resource utilization. The authors consider access to a shared storage space for storing VM disk images. In our approach, live migrations are executed without shared storage, avoiding network bottlenecks.

Beloglazov et al. [BB10b] focused on energy-efficient resource management strategies that can be applied to a virtualized data center by a cloud provider. The authors proposed the development of thermal, network and multiple system resources optimizations, embedded in a decentralized architecture of a resource management system for cloud data centers. The approach is based on live migration to dynamically reallocate VMs. The architecture consists of a dispatcher and global managers, allocating thermal, networking and workload information from local managers as well as running distributed reallocation decisions. Beloglazov et al. used a distributed version of a semi-online multidimensional bin-packing algorithm, focusing on the system's energy efficiency extension as well as considering quality-of-service (QoS) requirements. The approach has been evaluated using simulations, showing a reduced power consumption in combination

with assured QoS. But in contrast to our approach, its implementation as part of a real-world cloud platform is only planned for the future. This is a fact that nearly all papers related to energy-efficient resource management in cloud computing have in common. Therefore, increasing the potential for power savings in a real implementation of a cloud computing system, i.e. with realistic costs for VM live migration, is an interesting challenge.

### 4.2.6 Summary

In this chapter, virtual machine consolidation in IaaS clouds was presented. The approach is based on performing live migrations for VM consolidation, allowing more servers in a cluster to power down. In contrast to related work, the energy costs of live migrations including their pre- and post-processing phases are taken into account.

Virtual machine consolidation realize transitions between VM locations (CT 1) in IaaS clouds. To make virtual disks available at the destination server, an on-demand mirroring technique based on distributed replication block devices (DRBD) was used. During normal operation, all read and write operations on the virtual disk image were performed locally (local storage). Only when a live migration was initiated, a block device was mirrored to another host (synchronized storage).

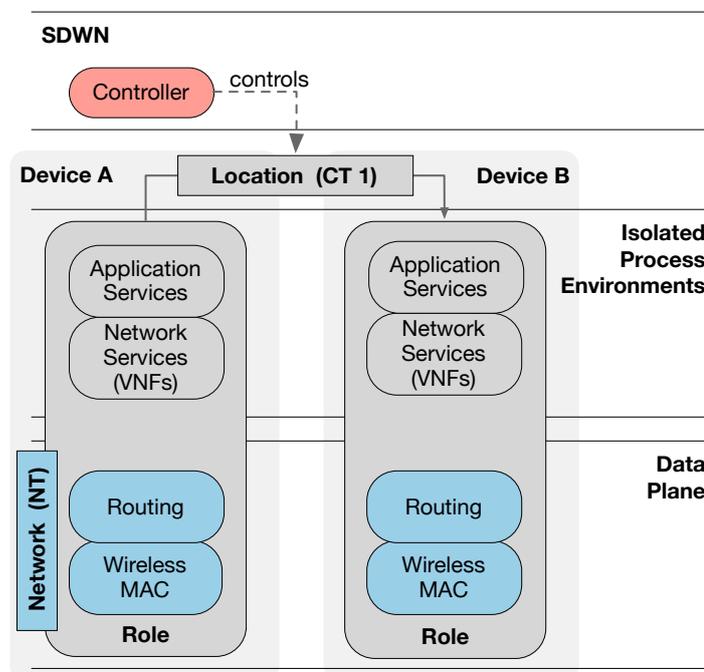
The proposed approach was evaluated experimentally using several short- and long-term tests based on virtual machine workloads produced with common operating system benchmarks, web-server emulations and different MapReduce applications. The results indicated that energy savings of up to 16% can be achieved in EUCALYPTUS environments.

### 4.3 Dynamic Role Assignment in SDWNs

In this chapter, a novel approach to dynamically assign roles to devices in a Software-defined Wireless Network (SDWN) is presented. First, its relation to transitional computing is described in Section 4.3.1. Design and the implementation issues are discussed in Sections 4.3.2 and 4.3.3. Experimental results are presented in Section 4.3.4. Section 4.3.5 discusses related work. The chapter is summarized in Section 4.3.6.

*Parts of this section have been published in [Gra+17].*

#### 4.3.1 Computational Transitions



**Figure 4.3.1** Computational Transitions within dynamic role assignment in SDWNs.

Figure 4.3.1 depicts computational transitions within the dynamic role assignment approach. Transitions between service locations (CT 1) are used to place services closer to mobile devices and to the users, in order to improve their latency. Furthermore, in order to save battery power of the involved devices, the arrangement of application and network services were combined with transitions between network mechanisms (NT). At the top of Figure 4.3.1, a controller is responsible to dynamically assign roles to devices and to control the performed transitions. On the devices, application and network services are executed within isolated (chrooted) process environments. At the bottom, within the data plane, the approach combines a wireless MAC mechanism like the 802.11 ad hoc mode with corresponding rules for IP packet forwarding, routing, and addressing.

### 4.3.2 Design

Roles support the heterogeneity of edge networks that consist of devices with different system architectures (e.g., x86, ARM or MIPS), network interface cards, CPU capabilities, and battery capacities. Central decisions based on the nodes' capabilities allow for changes in the topology and service placements when required. A role combines a node's wireless NIC modes with NFV and provides two basic functions:

**Topology Control:** Controls the wireless NIC modes (PHY/MAC layer) on a wireless device and their forwarding/routing capabilities (IP layer).

**Application-specific Services:** Run application-specific network and application services in an isolated environment.

Roles support the dynamics of an edge network, where physical topology changes occur frequently and where it is useful to adapt the physical topology, network and application services in a coordinated manner. For example, in public events with a larger number of participants (like a soccer game or a music festival), managed nodes can accept an offloading role to shift traffic from overloaded stationary access points to underloaded gateways. Furthermore, a managed node acting as an access point for unmanaged nodes might also be a proper location for a transparent HTTP proxy, providing caching and prefetching capabilities for WWW content.

The design of our approach follows the basic SDN principle to separate data and control plane. Figure 4.3.2 shows the proposed architecture to support dynamic role assignment. The control plane is realized via a central controller, while the data plane consists of stationary

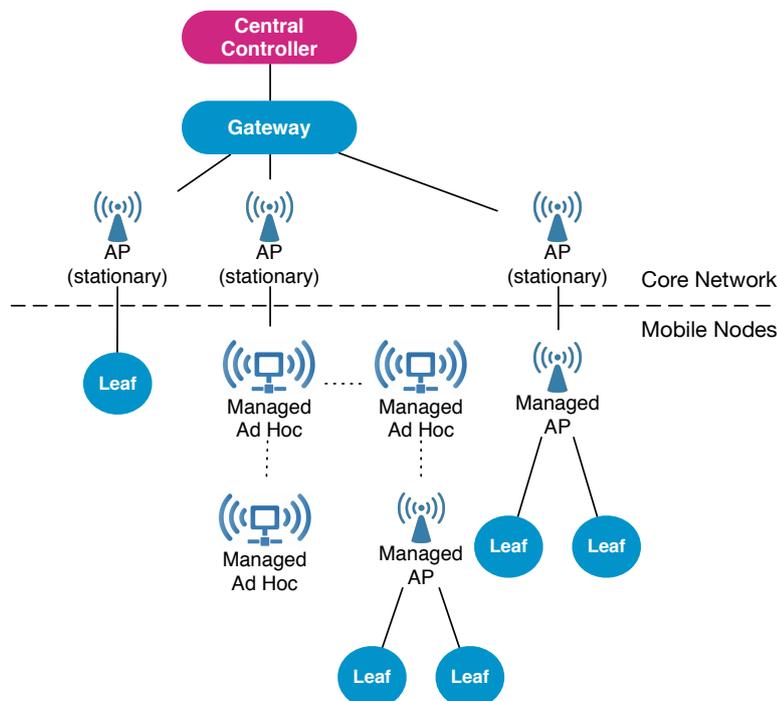


Figure 4.3.2 A network using dynamic role assignment.

access points, managed and unmanaged mobile devices. While stationary access points are part of the core network and act as gateways for the wireless devices' network traffic, a central controller instruments stationary and managed nodes. They are equipped with node controller functionality, allowing the central controller to modify the network configuration and to run services within an isolated environment inside a device under the control of an end user. Unmanaged nodes are unmodified wireless devices fully under the control of the end user. Thus, managed nodes are coordinated to act as relays or as network and application service providers for unmanaged nodes. Each managed node has one or more parents and a variable number of children. An in-band TCP connection serves as a communication channel for role assignments between the central and the node controllers. The central controller has two primary functions:

**Role Assignment:** Roles can be dynamically assigned to nodes during runtime using 'change\_role' messages. Currently, there are two roles: *leaf*, for network endpoints, *ap* for access points, and *ad-hoc*. In addition, there is also an auxiliary state *recovery* for those nodes that have lost connectivity to the network.

**Provisioning:** Certain forwarding mechanisms require that intermediary nodes between a network endpoint and the gateway must have their routes/flows provisioned dynamically whenever the network topology changes. In these situations, the controller will compute the necessary modifications and deploy them on the nodes.

### 4.3.2.1 Basic Roles

Basic roles provide a basic set of wireless NIC modes and forwarding mechanisms in a wireless network. They support elementary logical and physical topology transitions and can be extended by additional network and application services. Two wireless NIC modes are provided: infrastructure and ad hoc.

**Infrastructure Mode:** In this mode, a central node acts as access point and manager of the subnet. It supervises security policies and maintains a list of associated stations.

**Ad hoc Mode:** Here, the network operates on a peer-to-peer basis, where each station in the network is self-managed.

Four forwarding mechanisms are provided: bridging, pseudo-bridging, routing, and network address translation.

**Bridging:** This mechanism forwards data between a node's interfaces transparently using a native layer 2 bridge. Since it works on layer 2, both layer 2 and layer 3 protocols can traverse bridges, i.e., ARP and DHCP work natively without any helper applications. Furthermore, all devices operate in the same (sub)network, i.e., a single set of infrastructure servers, such as DHCP servers or gateways, maintains all nodes. However, in general it is not possible to bridge a wireless network interface running in managed mode. This can be addressed by the Wireless Distribution System (WDS), a non-standard extension to 802.11 that allows a 4-address frame format. However, WDS is not supported on all mobile platforms, and implementations can vary from one vendor to another to the point of incompatibility.

**Pseudo-Bridging:** A pseudo-bridge at layer 3 emulates the behavior of a layer 2 bridge. It appears to all other devices as a native bridge. However, in contrast to native bridges, there is no restriction on the types of interfaces that can be added, which gives this technology a far better compatibility. Since a pseudo-bridge is in effect a router acting as a bridge, it still has the limitations of a router. Most importantly, protocols that exist on layer 2 and/or rely on subnet-wide broadcasts cannot traverse routers. Two such protocols are ARP and DHCP. To allow these protocols to function in a network connected by pseudo-bridges, it is necessary to employ additional network services, listening for incoming layer 2 traffic, and forwarding it to the other relay interfaces.

**Routing:** Routing is performed in a network that is partitioned into a number of subnets. Each managed node is assigned to a separate subnet and acts as a gateway. Some protocols, such as DHCP, will not traverse network boundaries, which means that each access point will need to run its own infrastructure service. As with bridging, the routing approach uses native system utilities and does not have the same compatibility problems as bridging. On the other hand, subnets increase the management overhead within a network and create mobility/roaming and IP address distribution problems.

**Network Address Translation:** When using network address translation (NAT), the network is partitioned similarly to the routing mechanism described above. However, due to masquerading, this can be done in a simpler manner. The same address range can be used multiple times with different managed nodes. Besides, networks that are built in this way will be fundamentally static and hostile to roaming attempts.

### 4.3.3 Implementation

Our implementation relies on a Debian-based Linux on an ARM platform. Wireless NIC modes are switched using `nl80211 - 802.11 netlink interface tools`<sup>13</sup>. On top of this kernel interface, the `hostapd` user space daemon for access points or `wpa-supPLICANT` (IEEE 802.1X/WPA supplicant) is running. Since not all wireless drivers implement this interface, patched versions of `hostapd` can be used.

#### 4.3.3.1 Network Function Virtualization

Both network and application services use NFV on the controller to provide an isolated environment on the device, implemented as a process, running in a `chrooted` environment with root privileges. Additional software for network and application services is installed in an image file on the device's SD card and mounted as a loop device. Note that this environment is not yet capable of namespace separations for processes/network or `cgroup` resource management (CPU, memory or network). This might lead to security issues, and in case of too many different services, to resource contention. These problems are explicitly not addressed in our approach.

<sup>13</sup><https://wireless.wiki.kernel.org/en/developers/Documentation/nl80211>

### 4.3.3.2 Forwarding Behavior

Apart from access point or WPA supplicant network services, NAT and pseudo-bridging via `relayd` are provided as forwarding behaviors. Forwarding via NAT is achieved using native `iptables` masquerading, and a DHCP service (`dnsmasq`) is used for IP address distribution in the new subnet. The pseudo-bridging service is implemented via a userland daemon `relayd`, an all-in-one solution of the OpenWRT-Project to enable routers to perform wireless bridging.

### 4.3.3.3 Central and Node Controller

The communication between the central controller and the nodes is realized via an in-band communication channel. JSON-formatted messages are sent as newline-terminated strings via a persistent TCP connection. Both the controller and client software is implemented in Python3. Currently, the following message types exist:

**change\_role:** The controller sends this message to a node to tell it to change its role. Depending on the used forwarding mechanism, this message might include additional information that the node requires for the role switch. In the case of NAT, the node requires a specification of the subnet it is supposed to serve. A node using `relayd` does not require such additional data.

**check\_in:** Is sent when a node connects to the network for the first time. It informs the controller about the node's UID and MAC address.

Since IP addresses are variable, especially when using NAT, each node generates a UID on startup, to use for identification purposes. These UIDs are generated using the `uuid1` method from Python's `uuid` library. This incorporates both the devices' hardware address as well as randomness in order to generate a globally unique identifier.

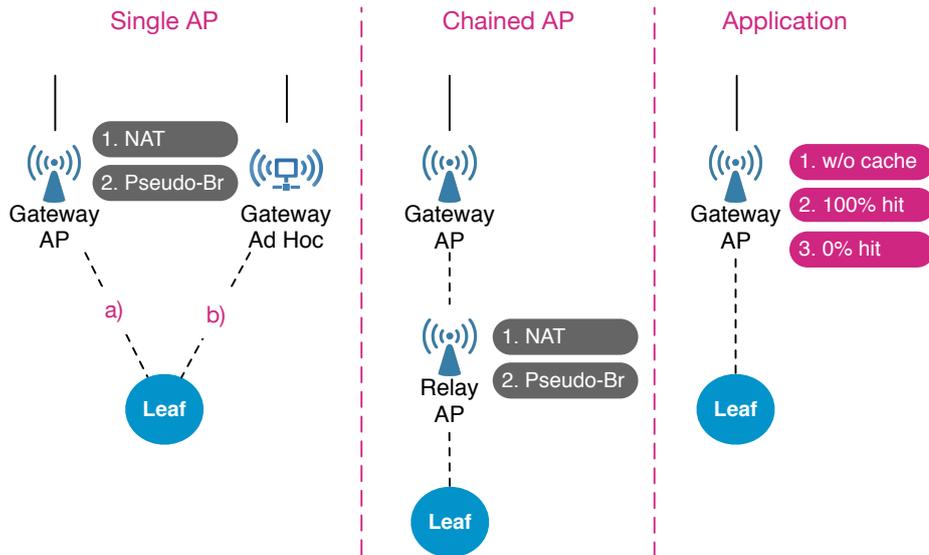
### 4.3.4 Experimental Evaluation

To evaluate our approach, power consumption, bandwidth, and round-trip time were measured. Figure 4.3.3 shows the three experimental setups that were examined.

**Single AP Setup:** One managed node either as access point (AP) or in ad hoc mode with one associated client. The managed node also serves as gateway to the public Internet.

**Chained AP Setup:** Two managed APs chained, the first with one associated client and acting as a wireless repeater. It is associated with the second managed AP's Wi-Fi, which also acts as a gateway to the public Internet.

**Application Setup:** One managed AP with one associated client. The AP also serves as a gateway to the public Internet. The basic role is extended with a transparent caching/forwarding web proxy (`squid`<sup>14</sup>).



**Figure 4.3.3** Experimental Setups: Single AP, Chained AP and Application.

The basic roles *bridging* and *routing* described in Section 4.3.2 are not considered in the experiments presented here.

All nodes are Raspberry Pi 3, model B. The node used as a wireless repeater is equipped with an additional Realtek RTL8188CUS USB Wi-Fi dongle.

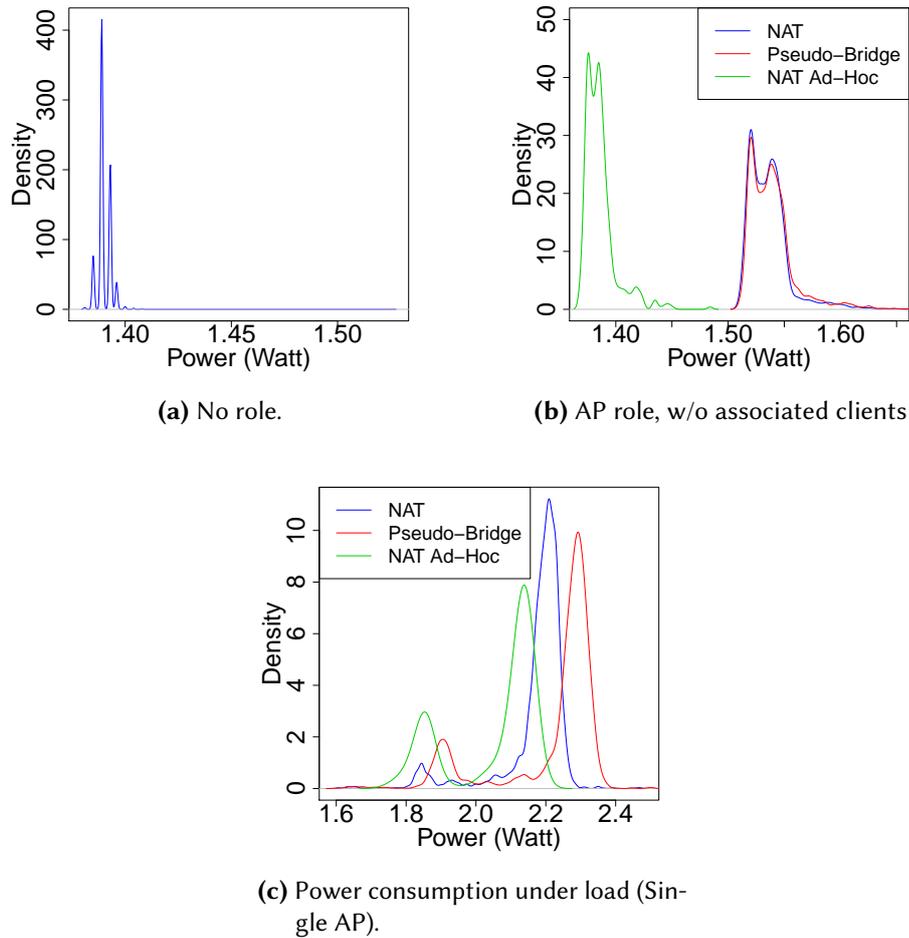
Bandwidth was measured using `iperf3` to saturate the network with the highest possible traffic, and 10 MB file downloads in the Application setup. Measurements were taken over a period of 6 minutes with a sampling frequency of 1 Hz, leading to 360 data points per measurement. In total, three measurement series were performed, leading to a total of 1080 data points for each measurement.

Power consumption was measured using the ODROID Smart Power meter. Measurements were 5 minutes long with a sampling frequency of 5 Hz, for a total of 1500 data points. Again, three measurement series were performed, which makes 4500 data points for each measurement altogether.

The round trip time was measured using the `ping` tool. Measurements were taken for both setups with the network being idle and under full load. For each measurement, 100 ICMP packets were sent at a rate of 1 packet/second, leading to a total of 300 data points over three series.

In the three experimental setups shown in Figure 4.3.3, the power measurements of the Relay APs were taken, and in the case of the Chained AP setup, the Gateway AP instead. Idle measurements were taken as a baseline. The remaining experiments were conducted under full load on the associated client. Density plots have been prepared using a statistical kernel density estimation with carefully tuned bandwidths.

<sup>14</sup><http://www.squid-cache.org>



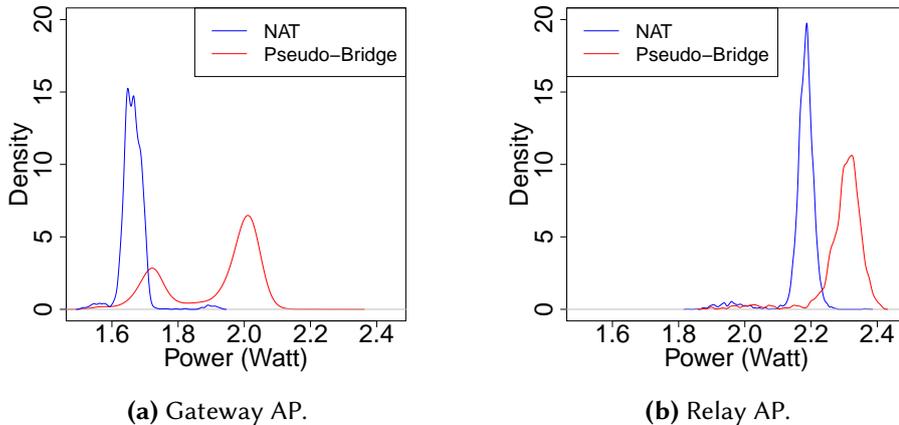
**Figure 4.3.4** Idle power consumption.

### Single AP Power Consumption

As a baseline, Figure 4.3.4 shows the power consumption of a node without a role and a node in AP role but without clients, respectively. Figure 4.3.4a shows two discrete power levels that the CPU occupies with a mean power consumption of 1.39 W.

Figure 4.3.4b shows NAT, pseudo-bridging, and NAT with ad hoc mode. When idling, the forwarding mechanisms seem to perform identically. The ad hoc mode consumes significantly less power. None of the systems has a higher power overhead. This means that any difference in power measured in the later stages is exclusively due to the way how traffic is handled and forwarded. The average power consumption measured at this stage is 1.54 W for both mechanisms, which represents roughly a 10% increase in power over the idle baseline. Most of this increase is due to the host apd network service running and keeping the Wi-Fi interface active.

Figure 4.3.4c shows a performance disparity between the NAT and the pseudo-bridging forwarding mechanism. Both curves are similar in shape; the curve for pseudo-bridging is shifted to the right. Both systems behave similarly, but the latter consumes more power. The mean



**Figure 4.3.5** Power consumption under load (Chained AP).

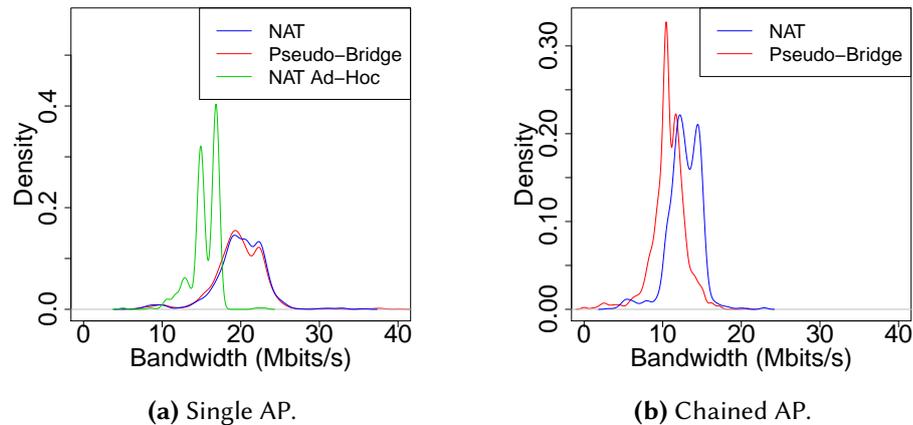
power consumption for NAT in this scenario is approximately 2.17 W, which is roughly a 40% increase over the previous scenario's 1.54 W. Pseudo-bridging has a mean consumption of 2.22 W, which is a 44% increase over the idle access point and 2% more than NAT. Ad hoc mode consumes approximately 2.05 W, which is an 32% increase over idle consumption, but a 5.5% decrease compared to NAT. While AP mode consumes more CPU cycles and also energy due to its management overhead, the ad hoc mode is significantly more energy-conserving.

### Chained AP Power Consumption

The power draw for Relay AP and Gateway AP was measured for a saturated network. The results are shown in Figure 4.3.5.

The curves for NAT and pseudo-bridging are different when compared to the previous setup. Gateway AP (Figure 4.3.5a) shows a single, though slightly bifurcated peak for NAT, meaning that the power draw is more or less uniform over time, while pseudo-bridging shows two distinct levels. The average of 1.66 W represents only a 7% increase over the basic AP. Pseudo-bridging, again drawing more power, shows an increase to 1.91 W, which is 24% more than the baseline and 15% more than NAT. This is by far the most significant difference measured between the two systems.

The situation gets even more interesting when looking at Figure 4.3.5b, showing the measurements for the wireless relay node. Again, pseudo-bridging consumes more power than NAT and the consumption of NAT appears to be more uniform. NAT of Relay AP consumed, on the average, 2.17 W, which is a 40% increase over the baseline and 30% more than the Gateway AP in the same scenario. The same is true for pseudo-bridging, which consumed 2.3 W, 49% more than the baseline and 20% more than the Gateway AP node. However, compared to the results of the Gateway AP from the Single AP setup, the values appear to be quite similar; 2.17 W versus 2.17 W for NAT and 2.3 W versus 2.22 W for pseudo-bridging. Thus, while the Gateway AP in the Single AP setup and the Relay AP in the Chained AP setup have similar power demands, the Gateway AP in the Chained AP setup consumes significantly less energy. Therefore, it appears that a node that services the client has an increased CPU and network load than a node that just performs a forwarding function.



**Figure 4.3.6** Bandwidth measurements.

### Bandwidth

Bandwidth measurements were taken for all setups. The results are shown in Figure 4.3.6. Similar to the power measurements, the curves are highly similar in the Single AP setup, as indicated by Figure 4.3.6a. Under these circumstances, NAT reached 19.9 Mbits/s, while relayd reached 19.7 Mbits/s, a negligible 1% difference. The NAT ad hoc measurement showed a significant lower bandwidth with 15.3 Mbits/s.

For Chained AP, shown in Figure 4.3.6b, the curves are rather flat and stretch out over a 5 Mbit/s interval, with pseudo-bridging being shifted towards lower bandwidths. NAT reaches an average bandwidth of 12.73 Mbits/s, which is 18% higher than the 10.76 Mbits/s reached by pseudo-bridging. This can be explained by the additional overhead introduced by relayd. NAT loses 36% of its bandwidth compared to Single AP, while pseudo-bridging loses 45%. To summarize, relaying the packets degrades the bandwidth significantly.

### Round Trip Time

Measurements for the round trip time were taken for all setups. Separate measurements were taken when the network was idle and under load.

Figure 4.3.7 shows the round trip times for NAT and pseudo-bridging. While the network was not experiencing any load, the round trip times average at 9.3 ms for Single AP and at 15.4 ms for Chained AP, with only slight deviations of less than 1% between NAT and pseudo-bridging. On the other hand, there is a significant increase of 40% between Single AP and Chained AP for both NAT and pseudo-bridging due to the overhead of an additional hop. Moreover, while Single AP shows no significant outliers above 200 ms, Chained AP has several of these up to 1500 ms (with a standard deviation at 135 ms in NAT and 142 ms with pseudo-bridging).

While the network was experiencing load, the RTTs for both Single AP and Chained AP setup are, as expected, very high: in NAT mode, 22.7 ms for Single AP and 46.6 ms for Chained AP (50% increase between both setups); with pseudo-bridging, 23 ms for Single AP and 80.7 ms for Chained AP (80% increase between both setups).

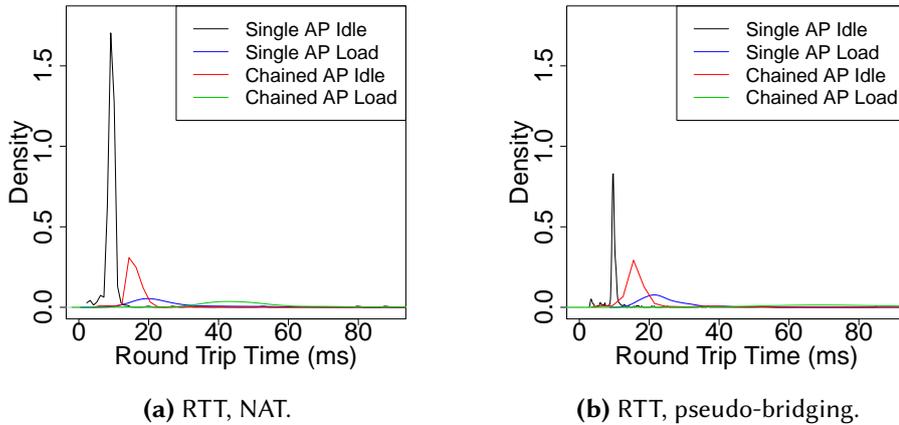


Figure 4.3.7 Round trip times.

Table 4.3.1 Caching Setup Measurement Results

Experiment	Throughput	mean Power	Energy
w/o cache	20 Mbit/s	2.33 W	39 mWh
100% cache miss	17.3 Mbit/s	2.46 W	40 mWh
100% cache hit	12 Mbit/s	2.21 W	36 mWh

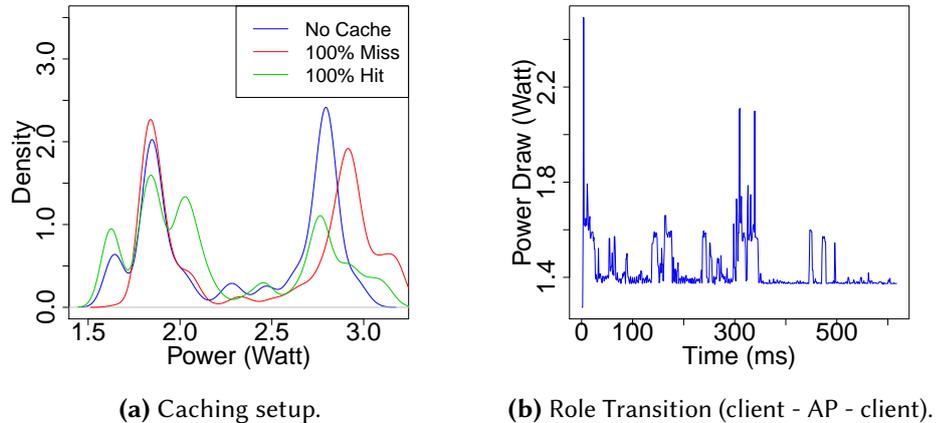
### Caching Setup

During this setup, 10 MB files are downloaded from an external HTTP server to a client. The traffic is routed via the Gateway AP node, running in a NAT role. Three measurements with a length of 60 seconds were taken: one without web proxy cache, one with web proxy cache and a 100% cache miss rate, and one with a 100% hit rate.

Table 4.3.1 shows the results. The highest energy is spent for the measurement without caching, the lowest energy is spent on the web proxy with 100% cache hit rate. On the other hand, the best bandwidth is achieved without caching. This is due to the high throughput of the external web server, which outperforms the ARM based Raspberry Pi 3 used in these measurements. Figure 4.3.8a shows the power consumption distribution of all measurements. The curves for no cache and 100% miss rate have a similar shape, with two distinct peaks separated by a valley. Therefore, there are two energy levels on which the node operates, one around 1.8 W and another at 2.8-3 W. When serving content from its internal cache, the node appears to operate at the higher energy level less often than when serving relayed data. The highest power draw can be observed for an empty cache, which is plausible, since in such a case the data needs to be both relayed and stored simultaneously.

### Role Change Overhead

Finally, the overhead of a role transition was measured. Figure 4.3.8b shows a node's power draw during a period where it was acting as an AP and in a period where it was a simple client. The switch from AP to client mode happens at approximately 300 ms where the graph shows two spikes, which coincide with the role change. The process took approximately 50



**Figure 4.3.8** Power consumption during caching and a role transition.

ms and generated a considerable increase in power consumption. Therefore, the role change time appears to be negligible, especially if one considers that the discovery time is primarily determined by the connecting device’s Wi-Fi Access Point scanning time, which is usually on the order of several seconds. The increased power draw can be explained by the fact that whenever network services are started or stopped, the need to load or unload a program generates an additional system load. This requires extra CPU work and results in a higher power consumption.

### 4.3.5 Related Work

Improving traditional cellular base station or Wi-Fi access point communication with device-to-device communication has been discussed in the context of 5G cellular [TUY14] and mobile ad hoc networks [CG14]. By contrast, our approach integrates network topology transitions as well as network service and application service placement within a single mechanism. Dynamically assigned roles enable a user to combine optimization approaches from different network layers in a novel way.

#### 4.3.5.1 Wireless SDN on Stationary APs

There are several approaches to centrally controlling and coordinating wireless access points, such as the CAPWAP protocol specification [CMS09]. A CAPWAP compliant network hosts a centralized controller responsible for user authentication, secure data transmission and limited provisioning of the access point using the Lightweight Access Point Protocol (LWAPP) [Cal+10].

Aziz and Vaidya [AV15] describe DenseAP, a system to centrally control transmission parameters, such as transmission power, minimum size and maximum size of contention windows as well as transmission rate in networks with densely deployed access points. Zubow et al. [ZZW16] present BIGAP, a fully 802.11-compatible architecture to provide high network performance and seamless mobility, which is achieved by fully utilizing the available radio spectrum and

a novel below MAC-layer handover. Bernados et al. [Ber+14] propose a general architecture for a Software-Defined Wireless Network (SDWN), incorporating both mobile internet (via 3G, LTE, etc.) and Wi-Fi networks simultaneously and allowing multiple service and infrastructure providers to share a single physical network. Suresh et al. [Sur+12] introduce the concept of a light virtual AP abstraction. These LVAPs can migrate between physical access points, physical access points in such a way that instead of client roaming between base stations with different BSSIDs, the LVAP will follow the device around the network. *OpenSWDN* is another SWDN approach developed by Schulz-Zander et al. [Sch+15a] to allow the execution of virtual middleboxes that were previously implemented on dedicated hardware.

However, these approaches address security, mobility and performance in networks with *stationary* access points, while our dynamic roles focus on *mobile* wireless nodes, specifically with different wireless NIC modes as specified by the 802.11 standards. The optimizations these aforementioned methods present, such as seamless handover and central control of transmission parameters, require a latency-constrained continuous packet stream of monitoring data to a central controller. Due to the in-band communication scheme used in our approach, introducing such mechanisms via relaying nodes would originate a significant overhead in energy consumption. A possible solution to this problem is the concept of local or "near-sighted" controllers [SSS14] running on relaying nodes. Near-sighted controllers could be implemented as a role (i.e., a particular network service) in our approach.

#### 4.3.5.2 Wireless SDN on Mobile Clients

Most of the approaches that address software-defined networking for NICs, such as the Data Plane Development Kit (DPDK)<sup>15</sup>, address high-speed packet processing in server environments and do not focus on ARM-based mobile devices and Wi-Fi. By contrast, Lee et al. [Lee+14] extend the SDN paradigm to mobile clients. Their proposal combines modifications of the stationary WLAN infrastructure, such as airtime scheduling with the capability of a local controller, which can instrument a local flow manager and packet scheduler. Meneses et al. [Men+15] extend SDN protocols with mobility management capabilities, where the mobile terminal can provide details about perceived connectivity targets and conditions, and the controller is able to enforce necessary changes to data flows all the way to the terminal node. However, these approaches are not capable of combining network and application services with topology information, due to the lack of a NFV implementation.

Syrivelis et al. [Syr+15] allow the collaboration of the users for sharing the idle capacity at the edge of cellular and Wi-Fi networks, trying to find the best Internet access routes. Users exchange roles of gateways, clients and relays, and connect to the Internet through multi-hop paths. Nevertheless, their implementation is based on wireless ad hoc mode only and relies on additional software on all participating clients. In contrast, our proposal allows both assigning an energy-conserving ad hoc role for managed nodes and an access point role that can also interact with unmodified clients.

---

<sup>15</sup><http://dpdk.org>

### 4.3.6 Summary

In this chapter, dynamically assigned roles in SDWNs were presented. It combined the centralized control of wireless Network Interface Controller (NIC) modes with Network Function Virtualization (NFV) to integrate network topology transitions as well as network service and application service placement within a single mechanism.

Dynamically assigned roles realize transitions between service locations (CT 1) within software-defined wireless networks. They allow network applications to place their network and application services close to users and their mobile devices. In order to save battery power of the involved devices, the arrangement of application and network services were combined with transitions between wireless NIC modes (NT).

The proposal was evaluated with respect to latency, bandwidth, and power consumption of the edge nodes. The experimental results showed significant differences in both bandwidth (up to 18%) and power consumption (up to 15%) for different roles and application services.

## 4.4 Summary

This chapter provided transitions between locations of functions, services, or modules of networked applications (CT 1) in near-\* computing environments. Mechanism transitions at several network layers were used to prepare and execute location transitions, further improving the ability of computing environments to adapt to dynamically varying demands while maximizing the efficiency of the infrastructure.

In particular, the following contributions were made:

- Virtual machine consolidation realize transitions between VM locations (CT 1) in IaaS clouds. VM live migrations are performed for VM consolidation, allowing more servers in a cluster to power down.
- Dynamically assigned roles realize transitions between service locations (CT 1) within software-defined wireless networks. Furthermore, in order to save battery power of the involved devices, the arrangement of application and network services was combined with transitions between network mechanisms (NT).

Both realizations allow computing infrastructures to apply dynamic service migration and role assignments in order to adapt to user mobility and demand variations. In addition, they reduced the power consumption of private IaaS clouds and devices in a software-defined wireless network considerably.

There are several areas for future work. Transitions between locations could be integrated into a commonly used SDN controller (e.g., ONOS<sup>16</sup>), in order to support the development of custom network applications that can be used in both wireless and wired SDN environments. Furthermore, on that basis, more types of algorithms could be developed to allow transitions between locations in more complex topologies.

---

<sup>16</sup><http://onosproject.org>



# 5

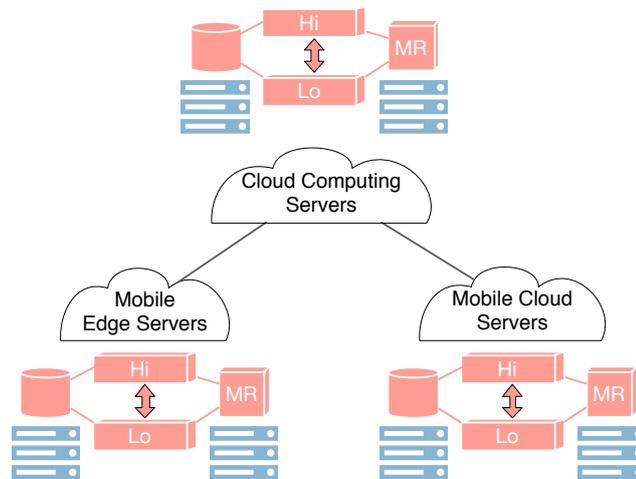
## Transitions between Implementations in Near-\* Computing

This chapter presents transitions between implementations in near-\* computing. In Section 5.1, two instances of this type of computational transition are motivated. Afterwards, these instances are presented in detail: Section 5.2 describes the novel approach of data sparsing for MapReduce, Section 5.3 presents the novel concept of opportunistic named functions and its realization. Finally, Section 5.4 summarizes the chapter.

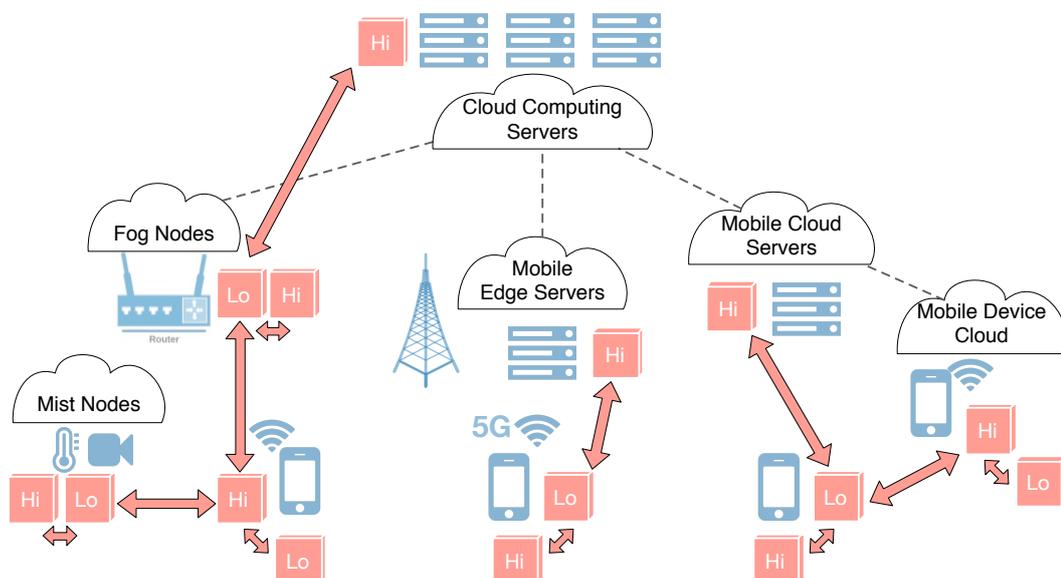
### 5.1 Motivation

In the recent past, shared data collected with mobile phone sensors like pictures taken by the device's camera or audio recordings coming from its microphone have been used for event detection tasks within different domains like sentiment estimation, person detection or recognition [AHO16]. In these scenarios, multimedia analysis tasks are applied to audio and video data, e.g., object detection and recognition, color signature detection, and feature extraction in images and videos, as well as speaker and speech recognition for audio recordings. For this purpose, several algorithms can be used with different runtime and resource consumption characteristics, but also accuracy levels.

Transitions between different algorithms, especially between different implementations of the aforementioned algorithms can be used to balance resource consumption, response times and the quality of the results in transitional near-\* computing. Figures 5.1 and 5.2 depict two instances of transitions between implementations in near-\* computing: data sparsing for MapReduce and opportunistic named functions. In Figure 5.1, data sparsing for MapReduce is shown. It adapts operators on data input streams for MapReduce tasks in near-\* data processing clusters. In Figure 5.2, Opportunistic Named Functions (ONFs) are depicted. ONFs are functions on named content that are processed near-user and near-network within an Information-Centric Network (ICN) on top of a Disruption-Tolerant Network (DTN). They allow opportunistic adaptations, i.e., the adaptation to locally available network- and battery-resources of mobile devices, like transitions between face detection algorithms.



**Figure 5.1 Data Sparsing for MapReduce** adapts data input streams of MapReduce tasks (MR, orange) in near-\* data processing clusters. It allows MapReduce tasks to perform transitions (orange arrows) between data stream operators (Hi/Lo, orange) at the distributed file system layer underneath MapReduce, modifying their power and runtime characteristics.



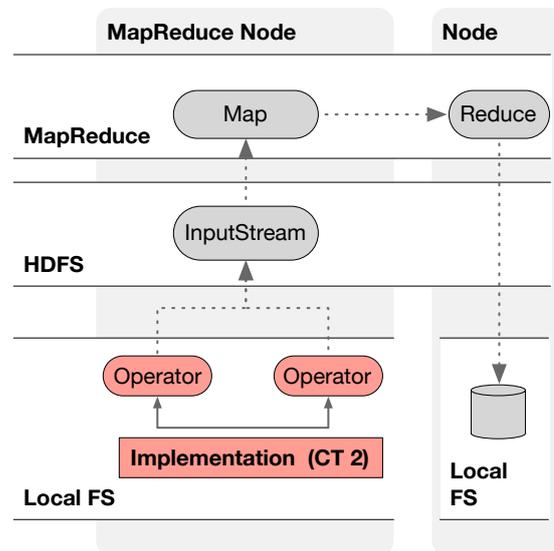
**Figure 5.2 Opportunistic Named Functions** allows functions (orange boxes) on named content to be processed within an Information-Centric Network (ICN) for near-\* computing environments. It is built on top of a Disruption-Tolerant Network (DTN) and allows opportunistic adaptations, i.e., the adaptation to locally available network and battery resources (orange arrows). For example, a transition between a resource intensive function (Hi, orange) and a function saving resources (Lo, orange) can be performed.

## 5.2 Data Sparsing for MapReduce

This section introduces the novel approach of data sparsing for MapReduce. It adapts input streams for MapReduce tasks in near-<sup>\*</sup> data processing clusters by performing transitions between data stream operators at the distributed file system layer in order to apply a novel concept: data sparsing with artifacts, a data reduction method for data processing applications that are robust to input variations, e.g., speech and image processing. At first, its relation to transitional computing is described in Section 5.2.1. Then, the design of our approach is presented in Section 5.2.2. The following Section 5.2.3 discusses implementation issues. The approach is evaluated in Section 5.2.4. Section 5.2.5 discusses related work. Section 5.2.6 concludes the section.

*Parts of this section have been published in [GHF15].*

### 5.2.1 Computational Transitions



**Figure 5.2.1** Computational transitions within data sparsing for MapReduce.

Figure 5.2.1 depicts computational transitions within the data sparsing for MapReduce approach. At the top, the MapReduce layer shows a MapReduce job that is split into map and reduce tasks. While the reduce task is kept unmodified, data sparsing is applied to the input data for map tasks. Conceptually, map tasks in the MapReduce framework process key-value pairs retrieved using HDFS input streams (HDFS layer). This abstraction sequentially reads HDFS blocks stored within the distributed file system. A single HDFS block, typically at least 64 MB in size, is stored within the local file system of a single HDFS (Local FS layer). Local file systems, however, typically have a block size of at most 4 KB. In data sparsing for MapReduce, the ability to perform operators on file system block level is introduced. Furthermore, transitions between operator implementations (CT 2) are performed in order to preprocess input data efficiently.

## 5.2.2 Design

In Section 5.2.2.1, data sparsing with artifacts is introduced. It is a novel approach to increase the energy efficiency of applications that are robust to input variations, such as speech and image processing. Its contribution with respect to energy efficiency is discussed in Section 5.2.2.2. Furthermore, its applications are discussed in Section 5.2.2.3, and its principle of data processing is described in Section 5.2.2.4.

### 5.2.2.1 Data Sparsing with Artifacts

Data sparsing with artifacts is inspired by dimensionality reduction methods that are often used in text and image processing, aimed at performing computations in a lower-dimensional subspace while preserving the properties of the higher-dimensional space. These methods are typically based on the theoretical work of Johnson and Lindenstrauss, who state - informally speaking - that any set of points in Euclidean space can be embedded into a lower-dimensional subspace such that all pairwise distances are preserved [DG03]. In particular, the random projection method [LHC06; LZ15] piqued many researchers' interest in the past. This dimensionality reduction method uses a random matrix to project points from a higher-dimensional to a lower-dimensional space. It has some interesting properties: The matrix can be constructed in an inexpensive way, since all its vectors are chosen randomly from the higher-dimensional space. Furthermore, the projection matrix can be chosen as a sparse matrix such that all elements are in  $\{-1, 0, +1\}$  with probabilities  $\{\frac{1}{6}, \frac{2}{3}, \frac{1}{6}\}$  [LHC06]. Empirical results on real-world text and image data have indicated that this method preserves the similarities between data vectors well, even with a moderate number of dimensions [BM01]. Random projections follow the approach of reducing the amount of data with inexpensive preprocessing based on statistical properties of the problem space. In a time of vast amounts of data processed in large data centers with high energy consumption, this approach is quite intriguing.

**Data Sparsing** utilizes this idea by reducing the execution time and thus energy consumption by processing a sparsed version of the original data. The term *sparsing* is borrowed from the mathematical term of a sparse matrix, a matrix in which most elements are zero. In data sparsing, data blocks are selected in a statistically independent manner and with a standard normal distribution on the basis of an application-dependent *sparse rate* with a given *block size*. It is similar to random sampling, where a sample is selected from a statistical population such that assumptions can be inferred from the sample to the population. However, in contrast to random sampling, data sparsing can not only omit blocks with a given probability, but also introduces a novel concept: Application-specific *artifacts*.

**Artifacts** are application-specific data blocks that replace blocks of the original data stream with an application-specific *sparse rate*. In computer science, the term *artifact* usually refers to an *undesired* alternation of data; in particular, compression artifacts stand for a loss in clarity in compressed image or audio files. In data sparsing, an *artifact* is a *desired* alternation of data within a data stream. Its purpose is, informally speaking, to indicate to an application that a given block *should be excluded* from the computation. With this mechanism, the result of the execution can be approximated *without modifying the implementation*. Thus, instead of sampling the data, i.e., omitting data blocks in order to reduce the amount of data to be processed, *artifacts* are introduced into a data stream. Artifacts are based on the following hypothesis:

Applications such as automatic speech recognition or face detection can usually exclude data sections very fast, if a detection within this section is very unlikely. For example, a block of zeros indicating a black rectangle within an image is excluded by a face detection algorithm at an early stage. Therefore, a block of zeros is an appropriate application-specific artifact for a face detection algorithm. On the other hand, a machine learning algorithm trained to detect single-colored shapes would exclude blocks of subsequent black/white colors, but would produce false positives for face detection-specific artifacts. Therefore, our basic hypothesis is: There are application-specific artifacts for each of these applications, and data sparsing with artifacts can decrease the execution time of this kind of applications.

### 5.2.2.2 Energy Efficiency

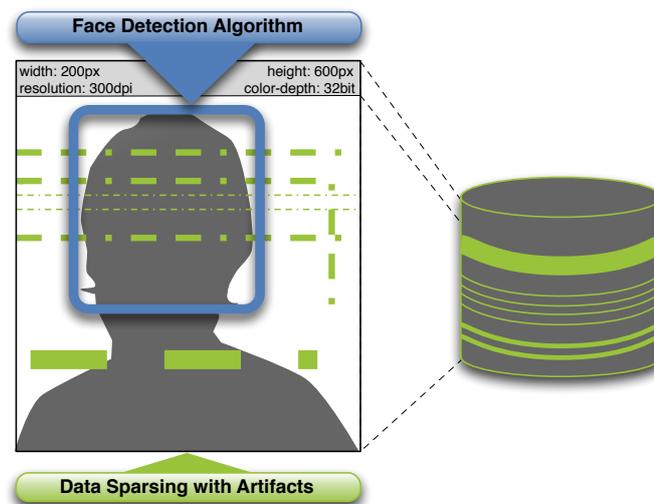
In general, the energy consumed by a system in the interval  $[t_0, t_1]$  is defined as the integral of power consumed in the domain of integration:

$$E(t) := \int_{t_0}^{t_1} P(t) dt. \quad (5.1)$$

The energy consumed by a system can be reduced by either minimizing the interval  $[t_0, t_1]$  (execution time) or  $P(t)$  for all  $t \in [t_0, t_1]$  (power consumption). Execution time and power consumption are often conflicting: For example, Chen et al. [CGK10] have examined I/O- vs. performance trade-offs in MapReduce applications. Depending on the compression rate, the use of compression for data transfers via a network is a way to conserve energy. But the compression rate is unknown beforehand, and experiments show that some compressed images increase energy consumption by increasing the use of the CPU. In contrast, data sparsing with artifacts is applied on the block level, i.e., blocks of data are manipulated without any knowledge about the data or preprocessing the data. We assume that data sparsing with artifacts neither increases power consumption nor decreases an application's execution speed, such that the reduction of execution time directly leads to reduced energy consumption.

On the other hand, active low-power modes, such as CPU dynamic voltage-frequency scaling (DVFS) or spun-down disks, reduce power consumption at moderate performance costs. Barroso et al. [BCH13] stated that typical server hardware is designed to operate most energy-efficiently at the maximum utilization level, and fine-grained active low-power modes can be used to gradually decrease the dynamic power consumption when the utilization level decreases (energy-proportionality) [BH07]. Conversely, disproportionately scaling down CPU voltage and frequency of a server might lead to a reduced energy efficiency. Using data sparsing with artifacts, execution time and thus energy consumption is reduced at moderate accuracy costs. Conceptually, the use of active low-power modes for hard disks, such as spinning down disks according to the sparse rate, would be a complementary method for data sparsing. Unfortunately, typical hard discs only provide inactive low-power modes, i.e., the device is not usable in this mode. On the other hand, DVFS may be applied for additional power savings, according to an application's CPU utilization. In these cases, additional power savings would be application-specific and independent of the data sparsing concept.

More energy savings could be achieved with modified disk controllers that can take advantage of sparsed sectors. However, we did not measure a significant effect of data sparsing on the power consumption of commodity hard disks and solid state drives. This is due to the design of



**Figure 5.2.2** The principle of sparsed data processing.

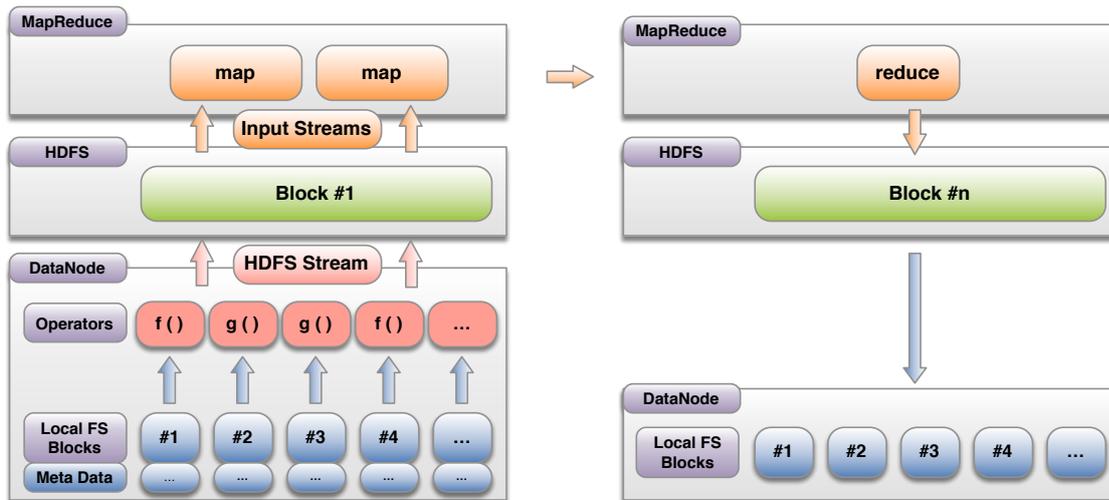
current disks that aim to achieve high throughput by making use of disk buffers and inexpensive read-aheads.

### 5.2.2.3 Applications

In principle, loss-tolerant applications such as image processing, sensor data processing, sound synthesis, as well as applications without unique answers, such as web search or machine learning can benefit from data sparsing. As stated by Esmailzadeh et al. [Esm+15], such applications can often tolerate lower precision or accuracy trade-offs. Furthermore, such applications process a growing amount of data: In contrast to former times when the majority of data produced was transactional data, the majority of data today shifts more and more to non-transactional data, produced by all kinds of devices [Nai15]. For example, social network providers like Facebook or Snapchat store petabytes of images and videos. An energy-conserving approach like data sparsing could help to improve the energy efficiency of their data centers. In contrast, applications based on transactional data such as log file analysis for accounting and statistics do usually not tolerate modifications of the original data and cannot benefit from data sparsing with artifacts.

### 5.2.2.4 Processing of Sparsed Data

Data sparsing with artifacts relies on blocks of bytes, i.e., sequences of bytes of a given length. This is due to the fact that the hardware architecture of servers is usually organized into virtual memory pages, hard disk sectors, packet sizes etc. If the *block size* is chosen as a multiple of a virtual memory page, data sparsing with artifacts can make use of page-table mechanisms to sparse the data. The basic principle of processing sparsed data is depicted in Figure 5.2.2: A program implementing a face detection algorithm is used to process a sparsed version of an image file. The sparse rate and the block size are specified such that the program is still able to process the image file successfully. The image meta-data, which is *sensitive* to data sparsing, is



**Figure 5.2.3** A Hadoop cluster including data stream operators at the local file system layer.

kept unmodified. This is an important assumption: Arbitrary text files may be sparsed without restrictions, but several other types of file contain sections that affect a program's control flow. Introducing an artifact into sensitive sections may result in runtime errors and unexpected program termination. Therefore, as shown in Figure 5.2.2, the raw data block containing the dimension and resolution of the image is kept unmodified.

In principle, data sparsing could be implemented within a user program (application layer) or within a special file system (file system layer). With respect to Figure 5.2.2, a realization on the application layer would read the original file from persistent storage and sparse it in-memory afterwards. But this design collides with our basic idea: Data sparsing should handle the software as a black box and keep the application unmodified. Therefore, data sparsing with artifacts is implemented on the file system layer. Potentially, this design allows energy savings even for applications where the source code is unavailable or restricted.

### 5.2.3 Implementation

In the following, the implementation of data stream operators at the local file system layer and transitions between them are described. In Figure 5.2.3, data processing in a Hadoop cluster is shown. In a Hadoop cluster, the MapReduce framework serves as frontend for applications. It runs on top of a Hadoop Distributed File System (HDFS). Files in HDFS are stored in blocks that are replicated between HDFS DataNodes. At the top of Figure 5.2.3, the two MapReduce stages are shown: (i) multiple map tasks on the left and (ii) a reduce task on the right. For both stages, the data processing pipeline is depicted. Mapper and reducer tasks operate on key-value pairs exclusively. The input specification of a MapReduce job declares an input splitter for byte-oriented input of an individual mapper, and a record reader that converts the byte-oriented input to key-value pairs, serving as input for a map operation (not shown in Figure 5.2.3). Splitters rely on HDFS data streams primitives that sequentially read from HDFS. HDFS file system blocks, however, persist within the local file system of HDFS data nodes. Typically, a single HDFS block is at least 64 MB in size. Local file systems, however, have at most a block

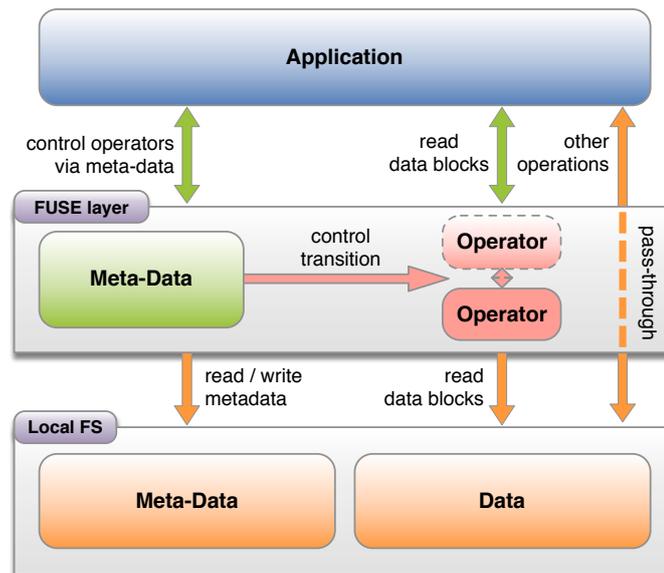


Figure 5.2.4 Implementation of Transitions between data stream operators.

size of 4 KB, corresponding to a typical block size of a hard disk or solid-state drive that is controlled by the disk controller. At that level, data stream operators are introduced: they perform byte-oriented operations on blocks of data within the file system of the OS. Finally, on the right side of Figure 5.2.3, the reduce task receives the combined and ordered key-value pairs from the output of the mapper task. It stores its output in the HDFS file system.

### 5.2.3.1 Local File System Modifications

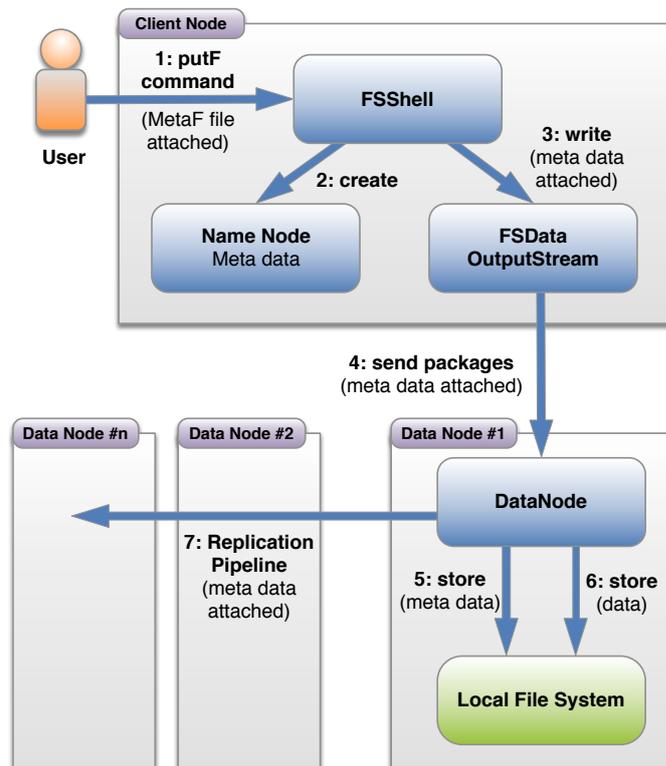
Data sparsing for MapReduce requires a mechanism to specify operations to be performed on file block level. Given that these operations have previously been defined, a processed version of this file can be delivered to the user process during a read system call. The original data is kept unmodified on the I/O device and can alternatively be accessed without processing via the local file system. The specification of the operations is stored as file system meta-data, so that all file types can be handled equally. In this implementation, transitions on data stream operators are realized by modifying the meta-data of a file.

Figure 5.2.4 shows the modifications of the local file system. At the top, an application transparently reads and writes data blocks from a file system in userspace (FUSE)<sup>1</sup> layer. Operators are specified and transitions between operators are controlled via file system meta-data.

For storing meta-data at the file system layer, extended user attributes are used, which is an extension of normal attributes of file system inodes that are supported by a growing number of Unix file systems, such as ext2-4. Our implementation is based on the FUSE-based open source file system `bindfs`<sup>2</sup>. `bindfs` mirrors already-mounted file system parts to another location, similar to a bind mount (`mount -bind`). However, in contrast to a bind mount, the implementation stores additional meta-data in extended attribute values. File systems like ext2,

<sup>1</sup><http://fuse.sourceforge.net>

<sup>2</sup><http://bindfs.org>



**Figure 5.2.5** Data flow of a FSShell command line putF command, specifying the operations to be performed on block level.

ext3 and ext4 store each extended attribute in a single file system block (usually 1 KB, 2 KB or 4 KB). A list of extended user attributes is used to store the meta-data, to bypass the single block size limit for meta-data that would significantly limit the maximum file size. Each of the extended user attributes has a fully qualified name in the form of `user.block_n`, where `n` is the number of the meta-data block.

The file system interface is not altered, and operations like `open()`, `close()` or `write()` are unmodified. `ioctl()` is reimplemented to provide read/write access to the meta-data. The specification of the input stream operator is stored with the `setxattr()` command of the underlying file system. The `read()` function relies on the operations specified with `ioctl()`, which is retrieved with the `getxattr()` function of the underlying file system.

### 5.2.3.2 HDFS Modifications

In contrast to a local file system, HDFS provides a non-mountable single virtual file system distributed over many server racks, which is not capable of extended attributes. It provides a separate file system name space. All files are accessed either via the FSShell command line interface, or the Java interface.

Figure 5.2.5 shows the HDFS data flow. In order to introduce transitions between input operators, the following design changes for HDFS are necessary: (i) Additional meta-data specifying the operations to be performed on the block level. (ii) A modified block data verification scheme in

order to avoid data integrity errors due to data stream operations. (iii) A modified user interface, i.e., a modified Java interface and commands for the FSShell command line interface.

The key challenge of integrating additional meta-data to mark data blocks is to store it without breaking the high-throughput optimization and the data integrity of the file system. Usually, a single HDFS NameNode is responsible for storing file meta-data such as the number of replicas, the block-ids of the blocks of a single file as well as the whole file system namespace. According to the HDFS architecture, DataNodes are only responsible for reading and writing HDFS blocks. Since HDFS runs on top of a low-level file system, it is a question of granularity on which layer data blocks should be annotated as processed. Our design uses the local file system meta-data on a DataNode, which is introduced into the HDFS data flow. This meta-data is specified by the user *per HDFS file*, translated to HDFS meta-data *per HDFS block* by the user client and transferred to the DataNodes, where it is stored persistently using local file system meta-data.

On the DataNode, there is still a need for HDFS block verification to ensure data integrity. This feature reflects the fault-tolerance of HDFS, in order to countervail hardware faults of commodity hardware. HDFS uses a two-way integrity check, which is modified in order to verify the integrity of the data, which has not yet been processed: (1) HDFS verifies checksums on receipt of the block. (2) The HDFS DataNode periodically triggers a checksum-based block integrity test to detect failures due to corrupt hard disk blocks. With respect to the user interface, the information about operations on block level is specified manually by the user in a MetaF file.

### 5.2.4 Experimental Evaluation

In the following, two use cases for data sparsing for MapReduce are evaluated: (i) face detection and recognition and (ii) speech recognition. Data sparsing is performed with a block size of 4 KB, while transitions between data stream operators were performed to allow varying sparse rates with different artifacts. 7 different sparse rates are used: 0.5%, 1%, 2%, 5%, 10%, 20% and 50%. The experiments are conducted in a Hadoop cluster, consisting of 1 NameNode and 2 DataNodes. The cluster size is related to the capacity of the EasyMeter<sup>3</sup> electricity meter that is used for the power and energy measurements. It is connected to a separate Ethernet network via the Multi Utility Communication (MUC) tool and polled at a 5 Hz rate. The servers have an Intel Core i7-4771 with 3.5 GHz, 32 GB RAM and 256 GB SSD and an idle power consumption of 55 W.

#### Use Case 1: Face Detection/Recognition

Several social-network or instant-messaging providers need to handle a vast amount of image files. An interesting use case deals with detecting and to recognizing faces within images uploaded by users, in order to provide better services to the users (e.g., suggesting potential friends to the user) or to their advertisers (to improve user-related ads). Therefore, a face detection and face recognition use case as a multimodal MapReduce application. The Hadoop MapReduce framework basically operates on input, output, and intermediate data structured

---

<sup>3</sup><http://www.easymeter.com>

	Yale	CIT	Random
Avg. file size	159 KB	34 KB	1058 KB
Avg. percentage	95.9%	70.5%	99.9%

**Table 5.2.1** Average file size and average percentage of modified file system blocks.

using key-value pairs. We use Mapper tasks to process the image with `OpenCV`<sup>4</sup> or `javafaces`<sup>5</sup> and Reducer tasks to store detected or recognized images.

For our evaluation of both face detection and face recognition, three types of databases are used: The Yale Face Database<sup>6</sup> (Yale), Caltech Faces<sup>7</sup> (CIT) and a random data set consisting of photos from `everystockphoto`<sup>8</sup>, `Flickr`<sup>9</sup> and Google image search<sup>10</sup> (Random). In total, over 700 images stored in JPEG format as well as BMP format are used as our test data. A face detector with high detection rates and low execution times as described by Viola and Jones [VJ01] is used, implemented with the Open Source Software `OpenCV`. For face recognition, the `javafaces`<sup>11</sup> implementation is used, which is based on `eigenfaces` [TP91].

Yale consists of gray-scaled images containing one face per image, 11 individuals with different facial expressions or configurations. It is often used by researchers for face detection and recognition tests. Similarly, CIT contains 27 persons with different lighting/expressions/backgrounds. In contrast to Yale, the location of faces in images is more variable and the images are larger. Random contains a large number of different images with larger file sizes.

Compressed JPEG and uncompressed BMP image files are examined. Since sensitive file headers need to be handled accurately, the first 4 KB block is kept unmodified. Additionally, since JPEG uses a termination marker at the end of a file, the last block is also stored without modification. Table 5.2.1 shows the average file size and the percentages of modified file system blocks per file.

Beforehand, an appropriate artifact for these algorithms must be selected. Simply omitting insensitive blocks from image files reduce the quality of the results significantly, because this leads to malformed shapes in an image without any chance of detecting any faces. Good results can be achieved with zero pages, because the JPEG parser interprets zeros as a termination symbol. Furthermore, the JPEG standard enables the user to insert line-by-line or blockwise restart markers as basic error-correction indicators for the JPEG decoder.

Figure 5.2.6 shows the experimental results. Quality denotes the number of detected or recognized faces.

**90% quality** is reached for the Yale data set at sparse rates of up to 2%; line-by-line restart markers achieve best results with up to 5%. For CIT, line-by-line markers achieve a quality

<sup>4</sup><http://opencv.org>

<sup>5</sup><http://code.google.com/p/javafaces>

<sup>6</sup><http://vision.ucsd.edu/content/yale-face-database>

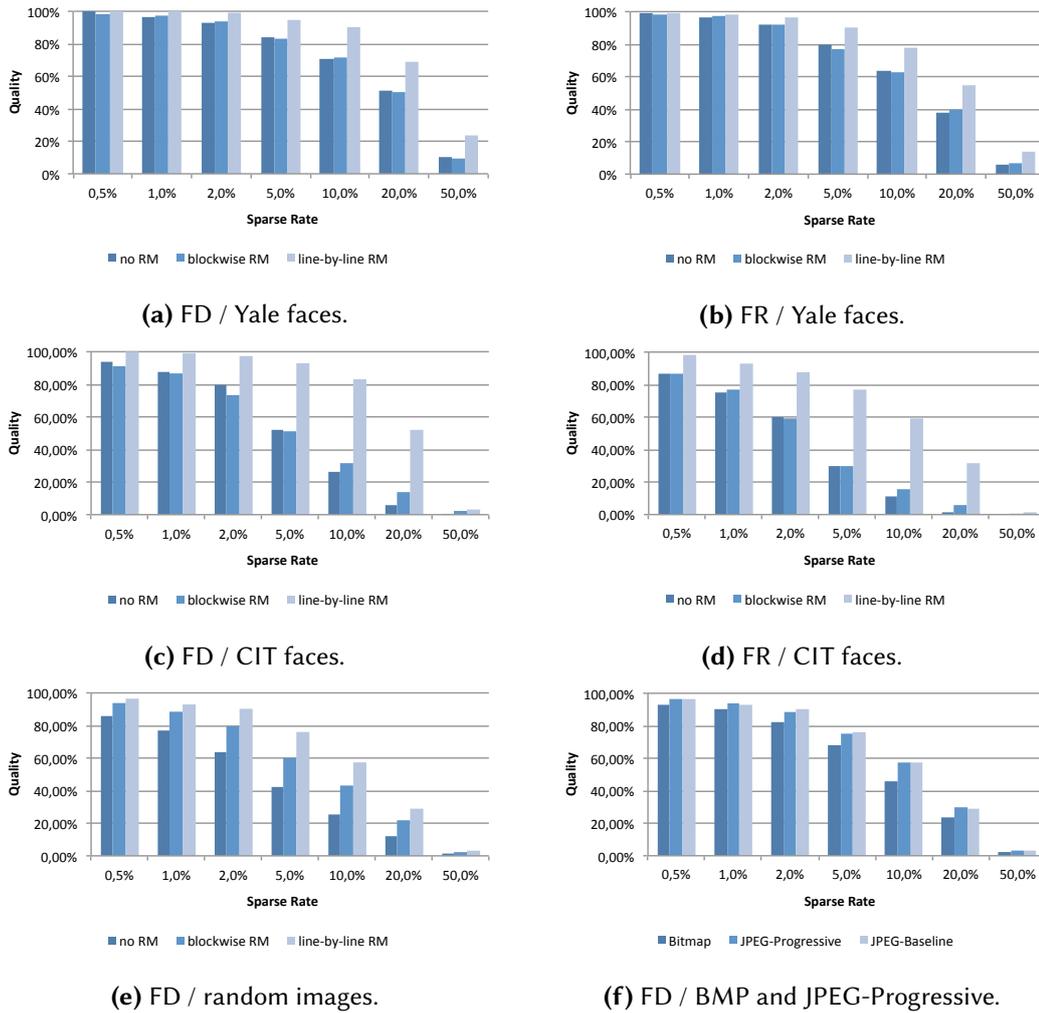
<sup>7</sup>[http://vision.caltech.edu/Image\\_Datasets/faces](http://vision.caltech.edu/Image_Datasets/faces)

<sup>8</sup><http://www.everystockphoto.com>

<sup>9</sup><http://www.flickr.com>

<sup>10</sup><http://images.google.com>

<sup>11</sup><http://code.google.com/p/javafaces>

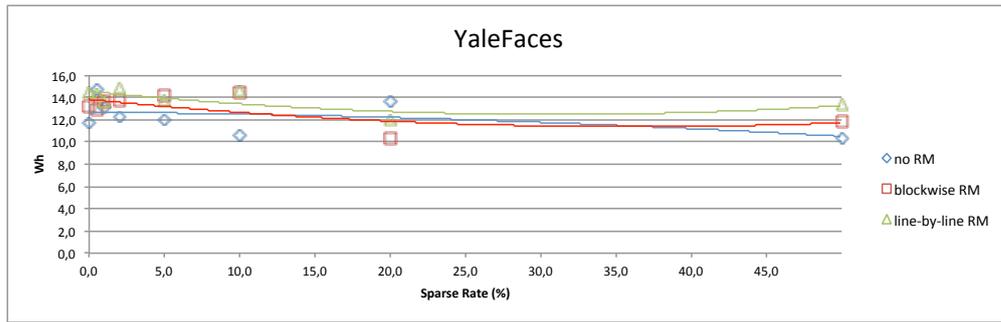


**Figure 5.2.6** Face detection (FD) and face recognition (FR) results.

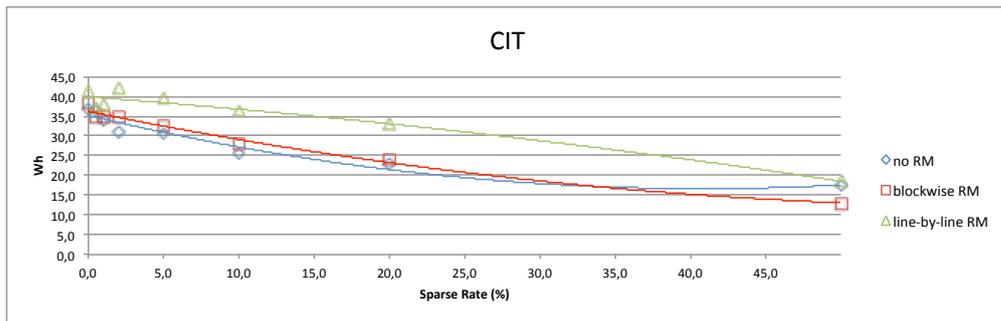
higher than 90% for sparse rates of up to 1%. Furthermore, for Random, only with line-by-line restart markers, 90% quality can be reached at a 1% sparse rate.

**75% quality** is reached for the Yale data set at sparse rates of up to 5%; line-by-line markers achieve best results with up to 10%. For CIT, 1% sparse rate is acceptable, and line-by-line markers achieve a quality higher than 75% for sparse rates of up to 5%. Furthermore, for Random, 75% quality can be reached at 1% sparse rate without restart markers; with line-by-line restart markers, up to 5% sparse rate.

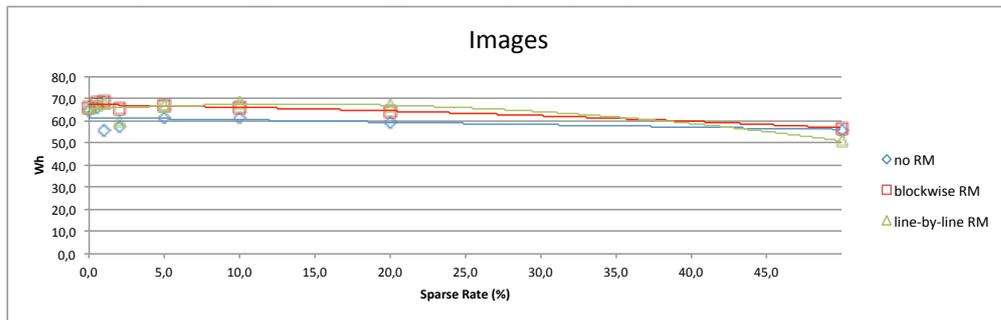
**The energy efficiency** for different sparse rates on different data sets is the following. For CIT without restart markers (Figure 5.2.7b), a sparse rate of 50% cuts the energy consumption in half. Within the range between 0.5% and 5.0% sparse rate, where 90% quality can be achieved, CIT and Random show a slightly reduced energy consumption (CIT: 17%, Random: 6%) without restart markers. Unfortunately, although restart markers achieved the best quality, they reduce energy efficiency. Without restart markers, the energy consumption baseline is better in all configurations. Due to the higher energy consumption of restart markers, at 90% quality, no energy savings are achieved. For 75% quality, data sparsing achieves 10% energy savings for CIT



(a) Face recognition energy consumption per sparse rate with Yale faces.



(b) Face recognition energy consumption per sparse rate with CIT faces.



(c) Face detection energy consumption per sparse rate with random images.

**Figure 5.2.7** Face detection and face recognition energy consumption results.

images, and 9% energy savings for the Random image database. As Figure 5.2.7a shows, the Yale images are not large enough to have a positive effect on energy consumption. The CIT data set is closer to realistic pictures, and it shows the best results with 1% sparse rate. Face detection on larger images as in 5.2.7c also shows an increase of energy efficiency within this range.

**In summary**, the reason for the drop in quality of the results in CIT and Random without restart markers is that the artifact introduced during data sparsing has a larger negative effect on the JPEG decompression algorithm than predicted. To sum up, the best results are achieved with line-by-line restart markers. Restart markers can increase the quality of the results, but lead to larger energy consumption. Blockwise restart markers introduce a data overhead of 10%, but do not lead to significantly longer execution times. Nevertheless, for 75% quality, data sparsing achieves 10% energy savings for face detection and face recognition on realistic image samples.

	PDA	AN4
Avg. file size	315 KB	178 KB
Avg. percentage	68,2%	44,5%

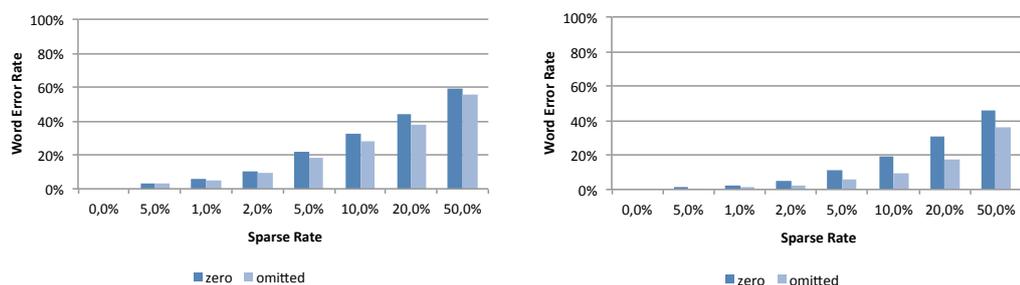
**Table 5.2.2** Average file size and average percentage of modified file system blocks.

## Use Case 2: Speech Recognition

Speech recognition has become a key feature for instant-messaging and mobile application providers. For example, mobile application developers can use cloud-based voice recognition providers like Wit.ai<sup>12</sup> to build voice-controlled applications. Audio files are recorded locally and sent to an analysis backend afterwards. Therefore, a speech recognition use case is evaluated below. For our evaluation, two types of databases are used: An alphanumeric database<sup>13</sup> (AN4), containing 1078 recordings of speakers spelling out personal information (such as name, address, telephone number, birth date) and a PDA speech database<sup>14</sup> (PDA), where the voice was recorded by microphones mounted on a PDA, and different speakers read about 50 sentences, resulting in 836 recordings. The PDA dataset contains Wall Street Journal news text, i.e., natural language with a large vocabulary. In principle, a large vocabulary means less accurate results, i.e., higher expected error rates. All data sets consist of .wav files with 16 bit sample rate.

For recognition, we used Pocketsphinx<sup>15</sup>, an open source speech recognition engine that recognizes words with an expected error of 20%. Pocketsphinx needs a phase of 3 seconds for calibration, which cannot be used for data sparsing. Thus, the file header and the calibration phase are kept unmodified. As shown in Table 5.2.2, this leads to a relatively low percentage of modified file system blocks per file.

Appropriate artifacts for this kind of algorithm are both zero and omitted pages. In case of .wav files, zero pages represent phases of silence, omitted pages interruptions within the recording. As Figure 5.2.8 shows, omitting pages gives slightly better results with respect to the word error rate. With the AN4 database, a 90% quality (10% word error rate) can be achieved with 10% sparse rate in case of omitting pages, and with less than 5% in case of zero pages. PDA



(a) Word error rate per sparse rate with AN4. (b) Word Error Rate Per Sparse Rate with PDA.

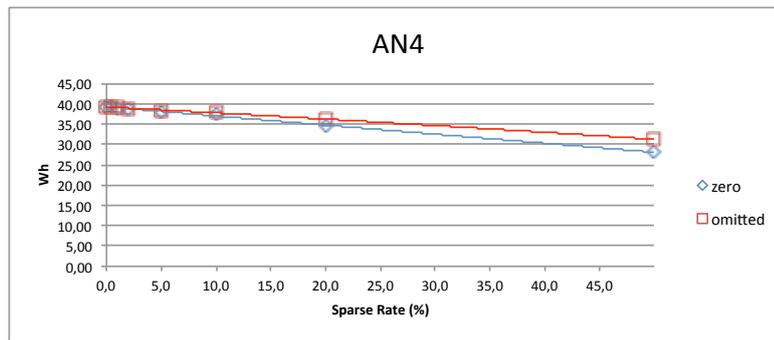
**Figure 5.2.8** Speech recognition results.

<sup>12</sup><http://wit.ai>

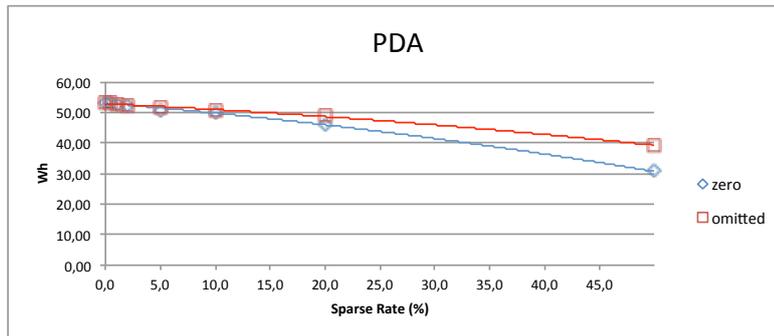
<sup>13</sup><http://www.speech.cs.cmu.edu/databases/an4>

<sup>14</sup><http://www.speech.cs.cmu.edu/databases/pda>

<sup>15</sup><http://cmusphinx.sourceforge.net>



(a) Speech recognition energy consumption per sparse rate with AN4.



(b) Speech recognition energy consumption per sparse rate with PDA.

**Figure 5.2.9** Speech recognition energy consumption.

has slightly worse word error rates: 90% quality can be achieved with only 2% sparse rate. Furthermore, a 75% quality (25% word error rate) can be achieved in AN4 with 20% sparse rate in case of omitting pages, and with less than 10% in case of zero pages. PDA can only achieve 75% quality with less than 10% sparse rate.

**The energy consumption** of both data sets decreases with higher sparse rates, as indicated by Figures 5.2.9a and 5.2.9b. 5% energy can be saved with 90% quality, while 8% energy can be saved with 75% quality.

## 5.2.5 Related Work

We discuss related work on two relevant topics: controlled approximation and approximate programming, and energy-conserving I/O-performance trade-offs.

### 5.2.5.1 Controlled Approximation/Approximate Programming

Several researchers try to find a trade-off between accuracy and energy consumption by using controlled approximation [BC10] and approximate programming [Esm+12]. Code perforation is a technique developed by Agarwal et al. [Aga+09] to increase the performance of error resilient applications. The approach basically performs loop approximations for performance improvements. It consists of an extended C/C++ compiler and a corresponding runtime environment, identifying a set of loops that are perforated according to a user defined acceptability metric.

Since this works automatically, it is not possible for a programmer to specify loops or parts of the program where code perforation should be omitted. The Green framework developed by Baek and Chilimbi [BC10] supports energy-conscious programming using controlled approximation, allowing function and loop approximation. To use the framework, a programmer has to indicate potential approximation points (e.g., function definitions and loops) using annotations and has to provide approximate versions of relevant functions.

The focus of architectural support for approximate programming is on voltage scaling for processing units and SRAM, floating point mantissa reduction and reduced DRAM refresh rates, to reduce the dynamic power consumption of CPU, cache and main memory. Approximate programming mainly targets highly utilized systems with high CPU, caches and main memory usage, but the general approach of trading accuracy for energy consumption is applicable to more cases, such as floating-point mantissa reduction [TNR00]. Sampson et al. [Sam+11] have proposed EnerJ, an extension of the Java programming language that enables a programmer to explicitly integrate approximation considerations into code. Esmailzadeh et al. [Esm+12] have introduced architectural support for approximate programming. In recent work, Esmailzadeh et al. [Esm+15] use artificial neural networks to learn how a region of approximable code behaves and automatically replace the original code with an efficient computation of the learned model.

These approaches target the same classes of applications as data sparsing. In contrast to data sparsing, they rely on specific programming languages, software modifications and custom hardware to increase energy efficiency. Data sparsing introduces artifacts at the data layer, without application layer modifications and with general purpose hardware. Furthermore, approximate programming handles bit flips and floating point mantissa reductions, resulting in a lower precision of computations. However, typical hypervisor and operating system resource management components operate on larger blocks. For example, a typical hard disk block or memory page is 4 KB. Data sparsing utilizes this fact by using memory page sizes as block sizes for artifacts. Furthermore, data sparsing is more than introducing errors into a byte stream, since it uses application-specific artifacts to indicate that a given block should be excluded from the computation.

### 5.2.5.2 Energy-Conserving I/O-Performance Trade-offs

GreenHDFS developed by Kaushik and Bhandarkar [KB10] is based on the concept of *hot* and *cold zones*. Rarely accessed files are stored in a *cold zone*, i.e., a set of servers with a low-power hardware design. Regularly accessed files, on the other hand, are stored in a *hot zone*, with higher utilization, performance requirements and power consumption. In contrast to our approach, GreenHDFS is based on system sleep states and low-power hardware at the cost of performance impacts and wake-up penalties. Chen et al. [CGK10] have examined I/O- vs. performance trade-offs in MapReduce. For some applications, the use of compression for data transfers via a network is a way to save energy. Nevertheless, a very low compression ratio of roughly 0.3 is necessary to achieve energy savings. For most data-intensive applications, this condition is not met. For instance, commonly used compression formats like MP3 and JPEG have compression ratios between 0.8 and 0.9 [CGK10].

### 5.2.6 Summary

In this chapter, data sparsing for MapReduce was presented. It performs transitions between data stream operators (CT 2) at the file system layer in order to apply a novel concept: data sparsing with artifacts, a data reduction method for data processing applications that are robust to input variations, such as speech and image processing.

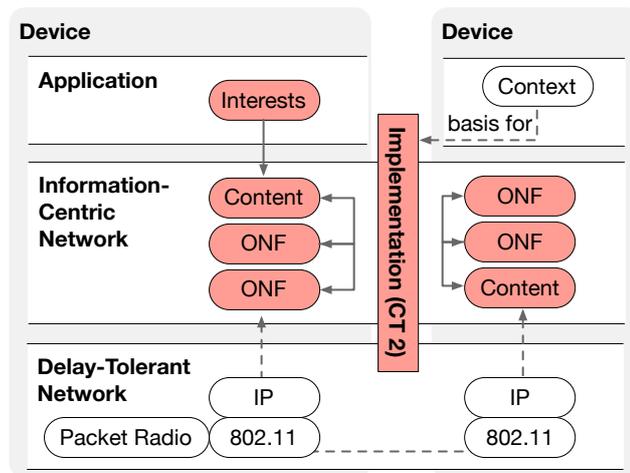
Data sparsing with artifacts is aimed at reducing the processing times and thus the energy efficiency of such applications while preserving the quality of the results by replacing a random subset of the original data with application-specific artifacts. Experiments with MapReduce-based face detection, face recognition and speech recognition algorithms show promising energy savings of up to 10% with moderate accuracy losses for different data sparsing rates and artifacts.

### 5.3 Opportunistic Named Functions

In this chapter, the novel concept of Opportunistic Named Functions (ONFs) is introduced. First, its relation to transitional computing is described in Section 5.3.1. The design and implementation of ONFs is outlined in Section 5.3.2 and in Section 5.3.3. ONFs are evaluated in Section 5.3.4, Section 5.3.5 outlines related work. Finally, this chapter is summarized in Section 5.3.6.

*Parts of this section have been published in [Gra+18a].*

#### 5.3.1 Computational Transitions



**Figure 5.3.1** Computational Transitions within opportunistic named functions.

Figure 5.3.1 depicts computational transitions within the approach of Opportunistic Named Functions (ONFs). At the top, a user or an application specifies interests in particular content, i.e., files that data producers are tagging with a unique name, or the output of an ONF. Interests in both named content and ONFs can be combined in a hierarchical naming scheme. These are then executed in the network on the fly, either partially or totally, in an opportunistic manner. During content delivery, nodes within the Information-Centric Network (ICN) in the middle of Figure 5.3.1 perform transitions between different variants of ONFs (CT 2): each node can either (i) perform functions on the original data or (ii) delegate the execution of the function to other nodes. ONFs can vary with respect to their runtime characteristics, i.e., to their resource and power consumption, but also to the quality of their results. Therefore, transitions between different implementations of ONFs can be performed on a single node based on locally available context information, i.e. on the number of interests in particular content known to a node, its battery lifetime, or device capabilities. The content delivery is handled by independent nodes that are connected in a Delay-Tolerant Network (DTN). On each node, different network links can be used for data transmission. On devices with interfaces to long-range, low-bandwidth, low-power radios like packet radio, ONFs can be used to preprocess content, i.e., to send only small amounts of extracted data via low-bandwidth links.

### 5.3.2 Design

In Named Function Networks (NFNs) [TS13; TS14; Sif+14], the network orchestrates function execution and caching, in an intelligent manner. For resource-constrained ICN-DTNs, the collaboration between nodes to efficiently execute functions and cache intermediate results is also highly desirable, but opportunistic communication introduces several problems: (i) communication is driven by content consumers *and* by intermittent opportunities, (ii) routing paths, computing nodes, and caches might be (temporarily) unavailable, (iii) global information about the network, i.e., its topology, statistics about different link qualities, and other information relevant for performing an orchestration is incomplete or sparse in the best case and unavailable in the worst case, (iv) mobile devices and sensors are resource-constrained and have limited battery capacity, and (v) these devices do not have a common architecture and might not be capable of performing complex functions.

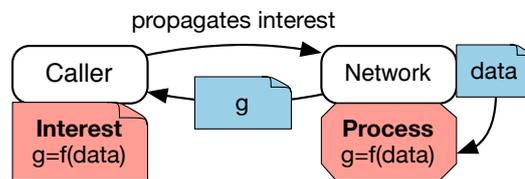


Figure 5.3.2 Basic ONF concept.

To address these problems, we propose *opportunistic* named functions (ONFs). Figure 5.3.2 shows the basic idea of our approach. A function  $f(data)$  is defined to be processed in the network. The node specifies a request to the network, similar to an asynchronous function call, where the node is the caller and the network is the callee. Each data producer in the network can either (i) perform the requested function on the original data or (ii) delegate the execution of the function to other nodes. In the best case, the request is processed completely within the network and the results are returned to the callee. In the worst case, the network delegates function execution on the produced data back to the callee, where  $f(data)$  is then applied.

Figure 5.3.3 illustrates in-network processing with ONFs in an ICN-DTN. Node A (upper left) stores cached content `/data`, which can serve as input for two kinds of function:  $g(/data)$  and  $f(g(/data))$ . In our approach, functions applied on existing content are specified using a naming scheme, where  $g(/data)$  is represented by `/data/g`. Furthermore, nodes signal their interest in the result of a function by declaring interest packets.

In Figure 5.3.3, green points (1-3) describe different points in time. At point (1), node B is in the range of node A that opportunistically transmits the cached content `/data` to node B. Node B, on the other hand, is aware of the two interests `/data/g` and `/data/g/f`, respectively, and after the reception of the data, node B decides how to process the content based on the known interests. In our example, the remaining battery of the node is low, so only one of two possible functions is applied and the result is stored in its cache. Then, node B moves to point (2), where it is in the range of more nodes, including node C. Since DTNs are based on the store-and-forward principle, all nodes in range receive the cached content. Since node C has a full battery, C decides to process `/data/g/f` and store the result in its cache. When node C arrives at point (3), the results of the function is transmitted to another node. To summarize, the data traverses the network from the content producer to the interested consumer, while it is processed on intermediate nodes.

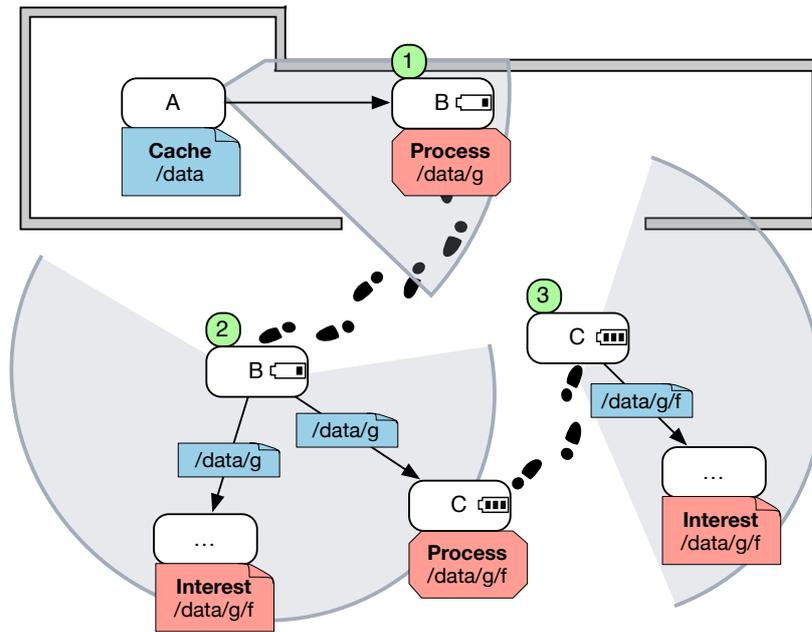


Figure 5.3.3 Processing of named content with ONFs.

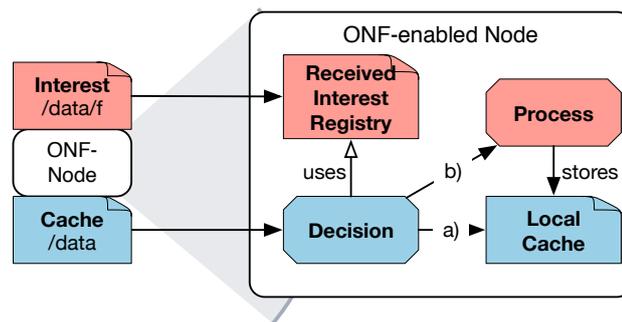


Figure 5.3.4 ONF Functionality.

Figure 5.3.4 shows the architecture of our ONF approach. First, a content consumer specifies his/her interest in content. This is expressed as an interest packet transmitted to an ONF-enabled next hop and then propagated opportunistically in the ICN-DTN network. An interest is specified as a human-readable hierarchical name that may refer to both content or functions. During interest propagation, at each hop, the received interests are registered locally. When new content is received from another node, the registered interests in specific content are used to execute opportunistic named functions. ONFs are applied based on locally optimal criteria as well as the priority of the registered interest. Afterwards, the results are cached locally. If a node has an intermittent connection to one or more other nodes, it sends the cached content in the order of its priority to the other nodes. This is explained in more detail below.

### 5.3.2.1 Interest Propagation and Registration

In existing ICN approaches (e.g., CCNx), a Forwarding Information Base (FIB) at each hop specifies the path on which an interest packet will be forwarded. Data packets traverse the network on the backroute of a received interest packet, stored in the Pending Interest Tables (PITs) at each hop. In ICN-DTN, the propagation of a content request is not coupled to forwarding/routing mechanisms. Decoupling function execution and routing allows to use different DTN routing algorithms. In our approach, interest and data packets travel on independent paths through the network. We use a bundle protocol on top of epidemic routing, detailed in Section 5.3.3.

Due to the loose coupling between content request resolution and routing/forwarding, a received interest might be outdated or the content might have already been delivered by another node. Received interests are stored in a Received Interest Registry (RIR). When a content consumer is no longer interested in specific content, a negative interest packet is sent by the client to the next ONF-enabled hop to propagate through the network, similar to an interest packet. The RIR is also used for basic bookkeeping purposes. It stores the last received interest or negative interest per user, and decides whether a received interest is newer than a stored interest. Therefore, an interest/negative interest packet needs a globally unique identification for the user and a sequence number.

### 5.3.2.2 Hierarchical Naming

Interests in ONFs are specified as human-readable hierarchical names that may refer to both content or functions, such as `/camera/persons/injured`. In this example, an operation is specified on the topic `/camera`, which represents all camera images taken from all participants. When a picture is taken with a camera of a mobile phone, a hook in the camera software library throws an event received by the ONF library. The ONF library then calls the name resolution engine described in Section 5.3.2.4, which is then responsible for performing the operation `persons/injured` on the camera image. Since ICN-DTNs do not use a global registry for specific files/filenames, our design of the hierarchical naming scheme does not allow to request a specific file like `/camera/image_001.jpg`. Instead of specifying a file request to the network, the content consumer specifies a topic and receives the corresponding content.

In addition to topics and topologies, the hierarchical naming scheme is used to express a *pipeline* of functions. For example, a content consumer can express his/her interest in all pictures taken at a specific location in the form of a filter on GPS latitude and longitude and a filter on all images containing a person `/camera/GPS_coordinates_50_8/persons`. Here, latitude and longitude are parameters for the filter function `GPS_coordinates`, encoded in the hierarchical name. When a content producer receives interests, then the node, based on the local context and its capabilities, decides which kind of and how many functions it applies on the produced content. It might store pictures under the name `/camera` and deliver them opportunistically to other nodes, where the GPS function is applied with the specified parameters. If the GPS filter is applied, the content is named `/camera/GPS_coordinates_50_8` afterwards. It either continues to detect persons, or it transmits the resulting image to another node able to perform the detection.

In case of multiple interests specified by one or more content consumers, the hierarchical naming is used to resolve *common tasks* that might be useful to serve as preliminary results for multiple interests *at once*. For example, in a disaster scenario, medical workers might be interested in all

pictures taken from injured persons, specified by interest `/camera/persons/injured`. Rescue teams might be interested in crowds of people, specified by `/camera/persons/crowd`, to organize supply or transportation for them. In these cases, the common interest in `/camera/persons` might be used for additional savings: either saving CPU cycles by forwarding the content of `/camera/persons` without detecting injuries or a crowd to both nodes, or saving transmission time by forwarding only pictures containing injuries or a crowd.

### 5.3.2.3 Alternative Names

ONFs rely on hierarchical and alternative names. While hierarchical names can be interpreted as subsequent executions (similar to an AND operator in several computer languages), alternative names introduce alternative function executions (similar to an XOR operator in several computer languages). This is especially interesting in scenarios where a slight reduction of the quality of results might be acceptable. For example, if a content consumer is interested in all images of fires specified by `/camera/fire`, it might not be possible to deliver a picture taken with a 16 megapixel camera to the content consumer, since interest and packet routes in an ICN-DTN are inherently non-deterministic and without guarantees. Instead, it is more useful to specify an alternative to that interest, with a higher possibility of a successful network traversal. For example, a user can specify the alternative `/camera/fire` or `/camera/wavelet_filter`, where wavelet transformations can be used for fire detection [Tör+06] that can run efficiently on digital signal processors, in contrast to a memory-intensive visual concept detection. In this example, alternative names are used to prioritize the interests with an XOR operator: if it is not possible to deliver results by visual concept detection, the network should use a wavelet transformation as a fallback. This concept is not limited to a single alternative. If multiple alternatives are specified, they are interpreted as an (ordered) cascade of fallbacks.

### 5.3.2.4 Name Resolution and Function Execution

Algorithm 1 is used to resolve names in ONFs. First, when new content is received, the device's context is used to get a list of functions that are available and applicable based on a device's context. Then, a loop computes the next functions to be applied on the content. This is used to allow pipelined execution of functions. Inside the loop, the resolvable interests are computed based on their naming scheme. For example, an interest in `/camera/fire` is applicable to the name `/camera`. Then, the corresponding functions are parsed using the longest prefix-match method that allows unambiguous name resolution.

The most important part is to decide which functions should be applied. This can be seen in the two functions `getPriorityMatches` and `getBestMatch`. The latter returns the functions that should be applied at this stage of the pipeline. For example, if both interests `/camera/persons/injured` and `/camera/persons/crowd` are specified, both functions are applied to the content with the name `/camera`. If there is an interest in `/camera`, the identity function is returned. More importantly, the function `getPriorityMatches` computes a total order of all functions, which is used to decide which functions should be applied. Therefore, each alternative name in an interest is handled in the order of its occurrence; the first name is handled with priority 1, the second with priority 2 etc. Then, the number of interests that can be served by a function according to its priority is used to compare each function, i.e., a function that serves two interests with priority 1 is preferred to another function that serves two interests with a lower priority. After the functions are executed, the loop is restarted at the next stage.

**Algorithm 1:** Name Resolution

---

**Input:** *name*: received content name, *F*: set of available functions, *I*: registered interests

```

1 Function parseFunction(interests, functions)
2   for i in interests do
3      $i.function \leftarrow \text{longestPrefixMatch}(functions);$ 
4   return interests.functions;

5 Function getPriorityMatches(parsedFunctions, functions)
6   for f in parsedFunctions do
7      $matches \leftarrow \text{priority of } f \text{ in } functions;$ 
8   return matches;

9 Function getBestMatch(interests, matches)
10   $result = \emptyset;$ 
11   $sorted\_matches = \text{sort } matches: m_1 > m_2 \text{ if } m_1 \text{ has more interests than } m_2;$ 
12  for m in sorted sorted_matches do
13     $result \leftarrow result \cup m.function;$ 
14    if all interests applied then
15      break;
16  return result;

17  $F' \leftarrow \text{get applicable functions from } F;$ 
18 for not  $F'.isEmpty()$  do
19    $I' \leftarrow \text{get resolvable interests for } name \text{ in } I;$ 
20    $functions \leftarrow \text{parseFunction}(I', F');$ 
21    $matches \leftarrow \text{getPriorityMatches}(functions, F');$ 
22    $M \leftarrow \text{getBestMatch}(matches);$ 
23    $name \leftarrow \text{apply each function in } M \text{ on } name;$ 
24    $F' \leftarrow \text{get applicable functions from } F;$ 

```

---

**5.3.3 Implementation**

ONFs have been integrated into the Serval Project [Gar+13a; Gar+13b; Gar+12] that is centered around a suite of protocols designed to allow infrastructure-independent communication based on DTN. The implementation of the ONF functionality in Serval is described below, as well as the applied functions in our scenario. Image transformation and face detection functions are introduced that can be used to trade between execution time, resource consumption and quality of results.

**5.3.3.1 Serval-based ONFs**

The Serval Project provides software for mobile devices to form a secure, self-organizing and fully distributed mesh network. Serval's store-and-forward DTN protocol (Rhizome) allows network operation in the absence of end-to-end connectivity and can run on top of a transport-agnostic Mesh Datagram Protocol (MDP). MDP can run both (i) on top of the IP protocol stack or (ii) on bare link layer protocols like packet radio. Serval Rhizome implements a simple

stateless flooding protocol running in user space. Since no guarantees for meeting other nodes are given, epidemic flooding of content to neighbors is used, providing fault tolerance and reliability at the expense of consuming resources. Data transmission is based on broadcast (announcements) and unicast (packet transfer), and the data can be transferred un- or encrypted and/or signed. These user space network layer mechanisms run with acceptable performance, both on common MIPS-processor based access point/router hardware and on low-end ARM smartphones [Gar+13a].

To provide basic ICN capabilities, we have designed an interface for specifying interests. Interests are specified in the form `/camera/face_detection`. Similar to other ICN-DTN instances [Mon+14], our implementation distinguishes between interest packets and content packets. They are realized with interest and content *bundles* in Serval Rhizome, which are transferred to other nodes without internal fragmentation or scattering. Furthermore, we have integrated generic hooks for handling incoming and outgoing payload by independent user space programs into Serval, called Serval Filters. These hooks allow user space applications to control (i) announcements of existing data to other nodes to forward interest and content bundles to other nodes at the sender, (ii) filtering of bundles at the receiver based on bundle metadata, and (iii) processing of received bundles. Consuming and producing bundles is handled by the same mechanism, i.e., a node can act as a content producer, an ONF node, or a content consumer at the same time.

An ONF-enabled Serval node maintains a Received Interest Registry (RIR), a name resolution component, and a function execution environment. Functions are implemented as executable binaries that reside either (i) in a designated folder inside the local file system, or (ii) as data bundles within the DTN network, allowing the distribution of (signed and encrypted) binaries via the network. When a new content bundle arrives at an ONF-enabled node, the name resolution algorithm decides on the basis of the content's name whether it is cached as it is or a function is applied (or both). In many scenarios, a pre-installed set of functions is available, such as official mobile warning apps distributed by governments and installed by mobile users.

### 5.3.3.2 Implemented ONFs

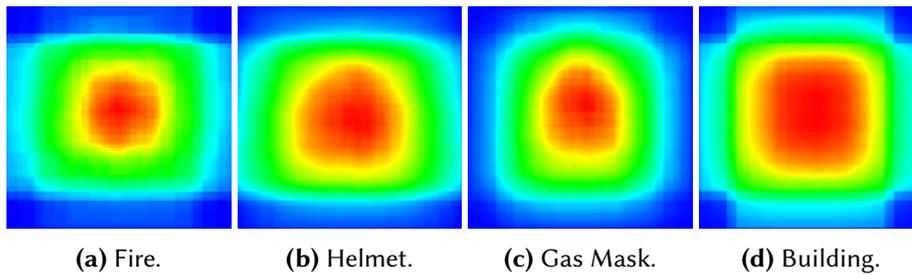
The implemented ONFs for our scenario are described below.

### 5.3.3.3 Image (Pre-)Processing

For basic image preprocessing purposes, we use the OpenCV<sup>16</sup> library that is available for multiple hardware architectures and software platforms. For black/white conversion of images, we use a basic thresholding operation to filter the intensity of each pixel above a threshold and assign to it a black value or a white value, respectively. Thus, we convert the original image to a 1-bit black-and-white image. To achieve grayscaling, we use an 8-bit grayscale operation integrated into OpenCV. Image scaling is performed by a resampling method using pixel area relations, which reduces the effect of moiré patterns in contrast to interpolation methods.

---

<sup>16</sup><http://opencv.org>



**Figure 5.3.5** Regions of interest for relevant topics.

#### 5.3.3.4 Smart Image Fragmentation

Another approach to trade result quality for transmission time is smart image fragmentation. It consists of two independent steps: (i) to fragment an image, where each fragment represents a region of the original image, and (ii) to prioritize and filter the fragments statistically, according to their relevance to detect a face or a visual concept.

This approach is related to the field of detection proposals [Hos+16] for machine learning approaches and makes use of the property that bounding boxes of interest are usually found within a specific region, and most importantly close to image centers [MV13]. Figure 5.3.5 illustrates this property by showing regions of high interest for relevant topics with red color and regions of low interest with blue color in a heatmap. Therefore, heatmaps were generated from bounding boxes provided by the ImageNet<sup>17</sup> image library that organizes images according to the WordNet<sup>18</sup> hierarchy, i.e., according to meaningful concepts, described by multiple words or word phrases. For each topic, a matrix with  $256 \times 256$  values was generated. These matrices represent the number of overlaps between the bounding boxes for a specific region of a normalized image. All bounding boxes for all detected objects of a topic were retrieved from the database and were scaled according to a normalized image size.

When image fragments are filtered according to these matrices, their relevance is determined *statistically* and not by inspecting a specific image. Assuming that fragmentation is performed during the compression phase and the matrices representing the statistical relevance are stored in memory, then this image fragmentation technique has  $O(1)$  runtime for a single memory lookup. Thus, smart image fragmentation can be applied on devices with small amounts of memory (KBs instead of MBs) and constrained computation capabilities. It possibly sacrifices some result quality for a reduced amount of data that needs to be transmitted.

#### 5.3.3.5 Face Detection

For face detection, we apply a two-stage approach. In the first stage, the Viola/Jones algorithm [VJ04] implemented in *OpenCV* is used, since it is a common out-of-the-box solution for face detection. It is comparatively fast [Che+15] and has a low false negative rate. Thus, as many faces as possible are detected quickly. However, it has a relatively high false positive rate that is acceptable if the algorithm is used as a pre-filter. In the second stage, *dlib*<sup>19</sup> is used to verify the

<sup>17</sup><http://image-net.org>

<sup>18</sup><http://wordnet.princeton.edu>

<sup>19</sup><http://dlib.net>

results. It is slower than the Viola/Jones algorithm, but in the second stage it operates only on small subimages that can be processed much faster. Additionally, *dlib* has a low false positive rate and a higher precision [Che+15]. Consequently, *dlib* can act as a validator for the results produced by the Viola/Jones algorithm. All parameters except the face sizes are independent of the used face detection algorithms. Thus, the algorithms can be replaced by alternatives, still benefiting from the rest of our optimizations.

### 5.3.4 Experimental Evaluation

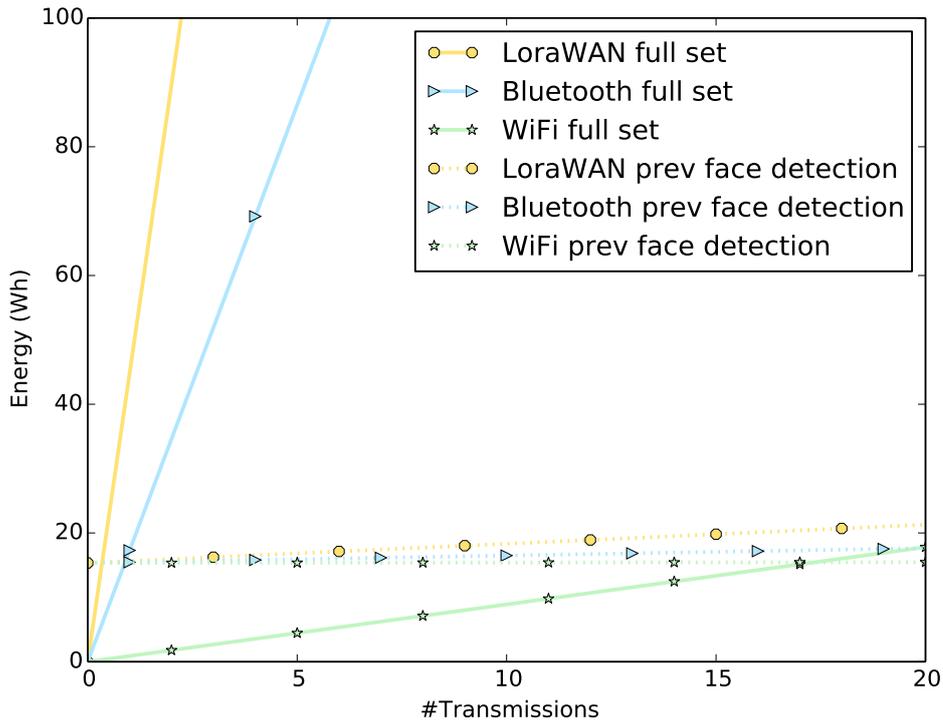
We present an experimental evaluation of ONFs below. In Section 5.3.4.1, image (pre-)processing, smart image fragmentation, and face detection are evaluated in terms of runtime and power consumption. In Section 5.3.4.2, these experimental results are used to simulate ONFs in different basic topologies as well as in a disaster scenario.

#### 5.3.4.1 ONF Measurements

To evaluate image (pre-)processing, smart image fragmentation, and face detection as examples of ONFs, we performed measurements on a Raspberry Pi 3, model B. This device is used as a reference for several devices with ARM-based CPUs, such as mobile phones. As a test image set, we used an existing image set that only contains images related to emergency scenarios [Lam+17]. It consists of 1,482 files, with a total size of 2.7 GB. It includes the following scenario specific search terms on an Internet image search engine: *Haiti earthquake*, *earthquake faces*, *earthquake people*, *disaster people*, *disaster faces*. Power consumption was measured using the ODDROID Smart Power meter with a sampling frequency of 5 Hz.

The experimental results show that black/white transformations of all images consume a total of 0.35 Wh (328 seconds at an average of 3.8 W), grayscaling consumes 0.31 Wh (296 sec at avg. 3.86 W), image resizing consumes 1.01 Wh (964 sec at avg. 3.79W), and face detection consumes 15.33 Wh (14,914 sec at avg. 3.7 W). While the average power consumption differed only slightly during this test, it shows that each face detection takes, on the average, 10.1 sec, with a standard deviation of 4.7 sec. Smart image fragmentation has been tested with pictures fragmented into matrices of  $128 \times 128$  images. In our tests, we reduced the total amount of transmitted data at the following rates: 5%, 10% and 25%. A face detection applied afterwards was also possible, but the reduction of 10% gave the most reliable results of 94.6% positive detections.

To estimate a possible tradeoff between the transfer of an image and an expensive computation like a face detection, we measured the power consumption of a LPWAN interface (a LoRaWAN device), a Bluetooth interface (via two Raspberry Pis) and a Wi-Fi interface (on the Raspberry Pi during a transmission of our test data set). We then compared these measurements of a data transfer with energy consumed when (i) a face detection was performed and only smaller images within bounding boxes of the detected faces were transmitted, and (ii) this smaller data set was transferred via different wireless links. Figure 5.3.6 shows both the energy consumed for the full data set as well as the data set resulting from a previous face detection. The x-axis shows the number of hops while the images traverse the network, the y-axis shows the energy consumption over time. Since the resulting data set is quite small compared to the original data (only 18 MB or 0.75%), the energy spent for transmission on links with lower bandwidth (LoRaWAN: 25 kbps, Bluetooth: 600 kbps) dominates the total energy consumption. While for



**Figure 5.3.6** Energy consumed for a transmission with and without a previously applied face detection.

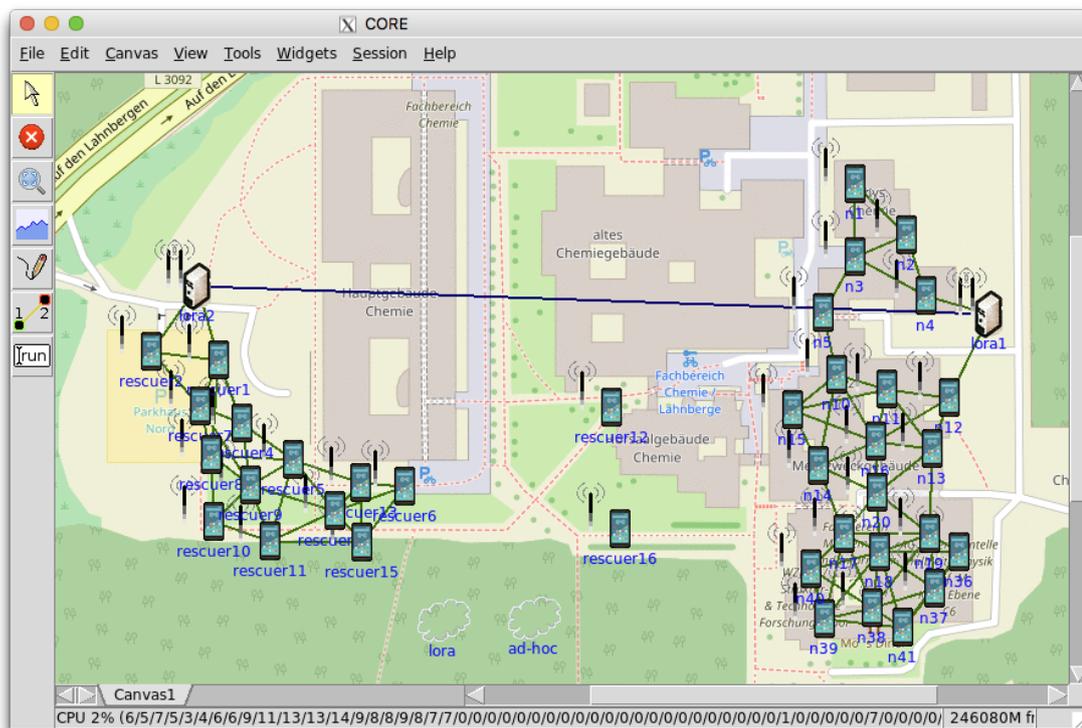
these links a face detection before the first transmission is useful, the break-even point for Wi-Fi is not reached before 16 hops. Applying an optimal strategy would require global knowledge about the network topology and its constituent links, which cannot be assumed in a network relying on opportunistic communication.

### 5.3.4.2 Network Topologies

We used the experimental results from the previous section to simulate ONFs in different basic topologies as well as in a disaster scenario. The described topologies were implemented using the CORE (Common Open Research Emulator) network emulation framework. CORE provides Wi-Fi access point and ad-hoc characteristics, traffic shaping, as well as simple network bridges for direct linking of certain nodes [Ahr+08].

### 5.3.4.3 Topologies

We examined three topologies: *Chain*, *Star*, and *Disaster*. The *Chain* topology consists of 64 mobile phone nodes that are connected pairwise in a chain. In this topology, the first node in the chain is defined as the content producer and the last one as the consumer. The *Star* topology consists of one middle node that acts as a gateway to all other nodes. The middle node is evaluated in two different configurations: (i) a lightweight router, only featuring basic data processing functions, and (ii) a powerful wireless router that is also capable of more complex



**Figure 5.3.7** Disaster scenario modeled in CORE. Rescuers approaching from the left, trapped people on the right.

functions like face detection. Figure 5.3.7 shows a screenshot of the *Disaster* scenario, a topology as it could be found in an emergency event. In the disaster site, 22 people are trapped and 15 rescuers are moving in the direction of the trapped people. To support the approaching rescuers, the trapped people document different aspects of the disaster using their mobile phones. A LoRaWAN link is established to enable basic text communication and to propagate the interests of the rescuers.

#### 5.3.4.4 Simulation Results

Based on these topologies, we performed several simulation experiments. The results are presented in Table 5.3.1. The  $\{Chain, Star, Disaster\}$  raw scenarios do not execute any functions, instead the images are spread to all nodes. They provide a baseline for comparison to  $\{Chain, Star, Disaster\}$  with ONFs. While the *Chain* scenario is evaluated with one content consumer and one producer, the *Star* scenario is evaluated in two different configurations: in the *Star-get* variants, all except one node act as content producers, whereas in the *Star-share* variant only one node acts as a producer and all the other nodes act as content consumers. The *Star* scenarios are further evaluated in two different configurations not presented in Table 5.3.1. In the basic configuration, the middle node acts as a simple router only providing inexpensive functions, such as smart image fragmentation. In the second variant, the router is also capable of applying the functions mobile phones provide, such as face detection. Finally, the ONF scenarios use our interest propagation and function application mechanism. These scenarios can be used to examine different metrics applicable to the proposed mechanism (e.g., interest propagation

**Table 5.3.1** Experimental Results  
Experimental results for all scenarios.

Name	avg( $t_c$ )	max( $t_c$ )	avg( $t_i$ )	$\sum$ bytes	$E_{trans}$	$E_{comp}$	$E_{sum}$
Disaster	19.41 s	214.22 s	3.68 s	371.2 MB	510.50 Ws	836.2 Ws	1346.72 Ws
Disaster raw	50.85 s	268.97 s	-	1.5 GB	2119.04 Ws	-	2119.04 Ws
Chain	27.42 s	101.08 s	7.76 s	488.4 MB	671.49 Ws	597.3 Ws	1268.79 Ws
Chain raw	79.56 s	389.77 s	-	2.8 GB	3881.70 Ws	-	3881.70 Ws
Star-get	8.33 s	20.11 s	0.27 s	131.3 MB	180.26 Ws	836.2 Ws	1016.48 Ws
Star-get raw	15.32 s	40.83 s	-	366.2 MB	503.62 Ws	-	503.62 Ws
Star-share	10.20 s	27.81 s	2.08 s	131.3 MB	180.26 Ws	836.2 Ws	1016.48 Ws
Star-share (lightw.)	11.01 s	31.04 s	4.08 s	354.8 MB	487.10 Ws	21.0 Ws	508.10 Ws
Star-share (router)	20.71 s	44.85 s	5.17 s	94.7 MB	129.34 Ws	784.5 Ws	913.90 Ws
Star-share raw	19.06 s	45.43 s	-	412.0 MB	566.91 Ws	-	566.91 Ws

time) as well as general metrics for comparison with the other scenarios (e.g., content delivery time). For the ONF scenarios, the interests are specified as follows: a first interest specifies a face detection on /camera, alternatively on fragmented or grayscaled images. A second interest specifies an image scaling performed on the original images or on a grayscaled- or a black/white-version of the image.

Note that the decision whether an ONF is executed or not depends on the devices' context. In our implementation, the expensive face detection functions are skipped if the battery level of a device is lower than 50%. Instead, grayscaling, fragmentation and black/white transformations are applied. The results of our simulations are shown in Table 5.3.1. The value  $t_c$  specifies the content transmission time, while the value  $t_i$  specifies the interest propagation time. Furthermore, the total amount of transmitted data and the estimated energy consumed (in Wattseconds) are listed. The latter is modeled by (i)  $E_{trans}$ , the energy consumed during transferring data at 10 Mbps via Wi-Fi (at 1.72 W, according to our experiments with the Raspberry Pi) and (ii)  $E_{comp}$ , the total amount of energy consumed by the applied ONF, according to the Raspberry Pi measurements.

Table 5.3.1 shows significant differences between *raw* and the ONF-based scenarios, although the same number of images were stored in /camera by the content producers. For the *Disaster* scenario, although no ONFs are performed and no faces are detected, the transmission energy spent by 1.5 GB of data is significantly higher (36%) than with ONFs. This is due to the data reduction of both scaling and face detection on the devices. Although more transmissions were performed (1475 compared to 702) due to multiple interests, the propagation time of the respective interest was also relatively low (3.68 sec), since the small interest bundles could be transmitted via the wide-range and low-bandwidth LoRaWAN. Using ONFs, the average content transfer time could be reduced by more than 60%.

In the *Chain* scenario, significantly more energy is consumed without ONFs (factor 3). In this long chain of nodes between consumer and producer, the latter scales images and detects faces according to the specified interest. Due to the high number of hops between consumer and producer, the interest propagation time  $t_i$  is rather long and the total amount of transmitted data is very high (488.4 MB vs. 2.8 GB). Therefore, the total energy consumption within this network is dominated by data transmissions. The content propagation time  $t_c$  is very high for

the raw case and can be reduced by again more than 65% on the average and almost 75% in the maximum case.

In the *Star* scenario, the interest and content propagation times are short due to the small number of hops between producer and consumer. *Star-get with ONF* and *Star-share with ONF* share the same behavior, since all ONF functions are executed on the producer nodes. Due to the small number of hops, the energy for function execution dominates the total energy consumed by this network. In such a case, ONFs deliver function results, but cannot utilize ONF for better energy efficiency. In contrast, a lightweight router in scenario *Star-get with ONF (lightweight)*, only featuring basic data processing functions is able to perform inexpensive tasks to reduce the total amount of data (14%) and the consumed energy (10.5%). In the more powerful router in scenario *Star-get with ONF (router)*, a face detection function is applied at the router. This is less energy-consuming compared to ONFs at the producer (10%), but it cannot utilize ONFs for better energy efficiency either. In the first two *Star-share* scenarios, content transmission times were reduced by a small amount, showing again a small advantage of this approach.

To summarize, the basic topologies show that ONFs applied at the producer are quite advantageous in scenarios with a high number of hops between consumer and producer, but in topologies with smaller numbers of hops, ONFs can be utilized for network congestion avoidance, but not for higher energy efficiency. We have shown that, even though ONFs require a rather high amount of time for computation, ONFs lead to shorter content delivery times. In a disaster scenario, on the other hand, a significant amount of traffic (factor 4) and energy (36%) can be saved compared to not using ONFs and flooding the network with raw images.

### 5.3.5 Related Work

In the past, in-network processing approaches have been proposed in a multitude of forms, ranging from Network Function Virtualization (NFV), which has been widely adopted by the industry, to approaches for clean-slate architectures in the future Internet, like the functional block-abstraction in the Autonomic Network Architecture (ANA) [Bou+10] and Named Function Networking [TS14] in ICNs.

#### 5.3.5.1 ICN-MANETs and ICN-DTNs

An early approach for name-based routing in broadcast media is the Listen First, Broadcast Later (LFBL) [MPZ10] algorithm. It relies on a design with minimal state-keeping and utilizes the broadcast nature of wireless networks in order to listen to broadcasted messages. Although ONF interests are not specified on topics and not on single data entities like files, ONFs share LFBL's approach of making decisions exclusively at the receiver, not at the sender. This inherently addresses the problem of *source mobility*, which usually requires additional mechanisms in ICN architectures (i.e., routing-based, indirection-based, or resolution-based approaches [ABS14]).

Several approaches feature mobile ad hoc networks for tactical operations, for example where group mobility and mobile nodes, and a hierarchical networking structure are utilized. For example, Oh et al. [OLG10] integrated a content-centric networking protocol suite that supports content addressing, repository, and distribution into a large MANET. The authors support two kinds of content: topic-based data (files) and spatial/temporal content (e.g., sensor values or

messages). However, this approach is driven by a hierarchical network topology that is relevant for soldiers and military purposes. In such networks of homogenous groups (like personnel, vehicles and satellites), hierarchically interconnected functions can be executed by design within the most appropriate group with respect to power and computing resources. In contrast, in our scenario we assume that the participating nodes in our ICN-DTN and the wireless links between them are more heterogeneous, so that a hierarchy of device types cannot be assumed. Based on delay-tolerant protocols [V+00; SPR05; HLC10], DTNs have evolved into ICN-DTNs. ICN-DTNs were recently used for message-based communication in disaster scenario notifications [Che+16a; Kim+15; Psa+14]. Here, naming schemes were utilized to specify a group of receivers (e.g., the police) and to prioritize the importance of certain messages (e.g., a SOS message is more important than a chat message). Within our ONF approach, roles and multiple receivers can also be expressed with the help of hierarchical names, i.e., specific function executions are only delivered to certain roles.

Monticelli et al. [Mon+14] assume that during a disaster, the network in densely populated areas is fragmented into smaller community networks. The authors leverage vehicles or mobile phone users wandering around to exchange ICN packets between isolated network fragments. In contrast to such mobile phone-only scenarios, radio technologies (e.g., Bluetooth, LoRaWAN, Wi-Fi, TETRA digital radio or satellite links) introduce more complex topologies of heterogeneous connections (between multiple classes of devices), which can be utilized by our ONF proposal with respect to transferring a smaller amount of data via high-range links at a low data rate.

### 5.3.5.2 In-Network Data Processing

In general, there are several use cases for and implementations of data processing in DTNs. Data processing and dissemination strategies in DTNs were broadly applied in a multitude of scenarios, for example, in connected drone swarms [Wu+17], surveillance camera networks [APO14], and smart city applications [Gia+]. These approaches solve application-specific problems to aggregate sensor values or to preprocess raw data. In contrast, ONFs are designed to exploit heterogeneous nodes and to process arbitrary functions, where they are appropriate.

The typical approach to perform face detection on mobile devices is to offload images to a server and run a face detection algorithm [Soy+12]. Although generator-powered servers might also be available in a disaster scenario, rescue teams and affected people cannot rely on it. In the domain of DTNs for disaster communication, preprocessing and delivery of medical images for healthcare workers was applied by Roy et al. [Roy+16]. In contrast to their application-specific implementation, ONFs can be leveraged to allow more applications running in parallel, and possibly benefit from each other by sharing a common set of preprocessed data.

Named Function Networking (NFN) was proposed by Tschudin et al. [TS14]. It was realized based on Content-Centric Networking (CCN) and used, e.g., as a method for realizing Content Delivery Networks (CDNs). In these approaches, partial or complete function execution is coupled with the ICN forwarding mechanism. In contrast, in ONF networks, routing and forwarding are decoupled from function execution. This allows functions to be applied opportunistically, hence we rely on *opportunistic* named functions.

Siflakis et al. [Sif+14] showed that preferential opportunism in the distribution of computation is possible for individual  $\lambda$ -sub-expressions in a program. In contrast, ONFs are based on

opportunistic communication and function execution. A prioritization of interests is reflected by alternative names.

Melvix et al. [MLP16] proposed a context- and tolerance-based forwarding strategy for IoT and 5G scenarios. Tolerances are used to tolerate longer delays, precomputed or approximate data instead of real-time sensor values. NFN is optionally leveraged to perform arithmetic functions on the data received from sensors. This is similar to our proposal of alternative names, but in contrast to our approach, it is specific to sensor data processing and aggregation.

Nguyen et al. [Ngu+17] presented a directional interest propagation mechanism for crowd sensing in NDNs. This approach distributes interest packets by only re-broadcasting them if the receiver is nearer to the area of interest than a latest sender. Traveling interest packets are minimized and the network load is reduced. In contrast, we distribute the interest packets to all participants and reduce the data size of the results by applying our ONFs.

### 5.3.6 Summary

In this section, Opportunistic Named Functions (ONFs) in Information-Centric Disruption-Tolerant Networks (ICN-DTNs) were presented. Users specify their interests in particular content and/or application-specific functions that are then executed in the network on the fly, either partially or totally, in an opportunistic manner. Opportunistic named functions rely on user-defined interests and on locally optimal decisions based on battery lifetimes and device capabilities.

ONFs realize transitions between named functions (CT 2) within ICN-DTNs. This allows opportunistic adaptations, i.e., the adaptation to locally available network and battery resources.

Experimental results showed that opportunistic named functions reduced network congestion and improve battery lifetime in a network of battery-powered sensors, mobile devices, and mobile routers, while delivering and processing information.

## 5.4 Summary

This chapter provided two instances of transitions between implementations (CT 2) in near-\* computing. This type of computational transition was used to balance resource consumption, response times and the quality of the results in image and audio data processing use cases.

In particular, the following contributions were described:

- Data Sparsing for MapReduce leverages transitions between input stream operators at file system level (CT 2).
- ONFs realize transitions between named functions (CT 2) within ICN-DTNs. It allows opportunistic adaptations, i.e., the adaptation to locally available network and battery resources.

These realizations reduced the power consumption of MapReduce processing clusters and saved network bandwidth in an information-centric delay-tolerant network (ICN-DTN) considerably.

There are several areas for future work. In addition to adaptations of data input streams of MapReduce tasks, which is a batch-processing system, transitions between input streams could be applied to data processing engines like Apache Spark<sup>20</sup>. Furthermore, incentive mechanisms to perform Opportunistic Named Functions locally for a global benefit could be explored, possibly by leveraging a game-theoretic approach.

---

<sup>20</sup><https://spark.apache.org>



# 6

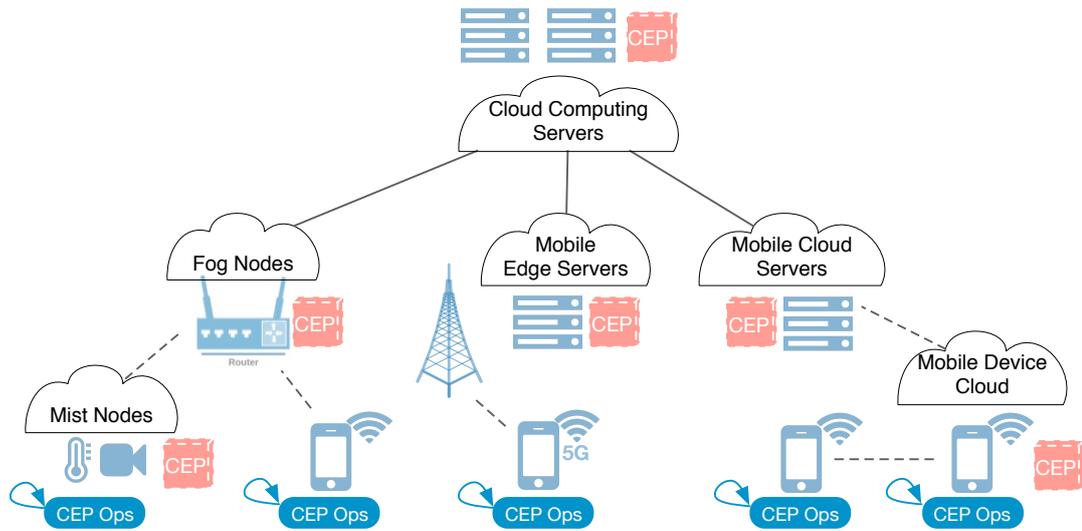
## Transitions between Modes in Near-\* Computing

This chapter presents transitions between modes in near-\* computing. In Section 6.1, two instances of location transitions for near-\* computing are motivated. Furthermore, these instances are presented in detail: Section 6.2 describes the novel approach of multimodal Complex Event Processing (CEP), Section 6.3 presents reactive programming for Wi-Fi firmware. Finally, Section 6.4 summarizes the chapter.

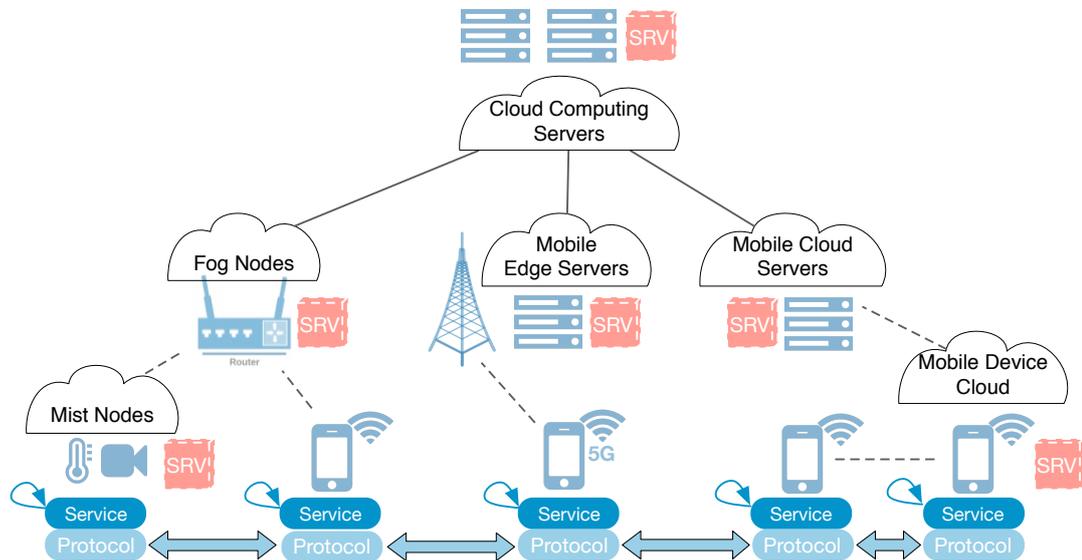
### 6.1 Motivation

Mobile phones, tablets, wearables, and battery-powered Internet-of-Things (IoT) devices offer exciting possibilities to create new edge and fog computing services that make use of the context of the device [Bel+12], such as user activities, location information, environmental conditions, operating system events, and network traffic, to preprocess data before transmitting it to cloud backends. However, today's mobile hardware/software architectures offer more capabilities for computations near-user, near-network, near-device, and near-data. System-on-a-Chip (SoC) designers have integrated low-power co-processors/accelerators, such as asymmetric multiprocessor architectures (AMPs), digital signal processors (DSPs), and sensor hubs. Transitions between modes leverage the full potential of these architectures by performing parts of network- and application services on-device 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).

Figures 6.1 and 6.2 depict two instances of transitions between modes in near-\* computing: multimodal CEP and ReactiFi. They complement existing architectures that already integrate various network- and application services. In Figure 6.1, multimodal CEP in near-\* computing is shown. Multimodal CEP automatically breaks up CEP queries expressed in a high-level language and selects the most suitable execution mode for the involved CEP operators, allows irrelevant components of a mobile device to be turned off or set to sleep mode, and avoids unnecessary CPU cycles, such as context switches and memory copy operations. In Figure 6.2, ReactiFi in near-\* computing is shown. Reactive programs can be used to monitor radio links, process frames and packets, and detect higher-level events to trigger actions in Wi-Fi firmware, such as switching communication modes, modifying Wi-Fi frames, or sending IP packets to a remote controller. On this basis, this approach enables reactive programs specified in high-level programming language to reconfigure the network protocol stack from within the Wi-Fi firmware.



**Figure 6.1 Multimodal Complex Event Processing (CEP)** in transitional near-\* computing. Existing architectures already integrate CEP close to the network’s edge (orange). Multimodal CEP is a complementary approach to access and process event streams even closer to the user, to the data, and to the network. Therefore, transitions between different modes of CEP operators (blue) are performed on-device.



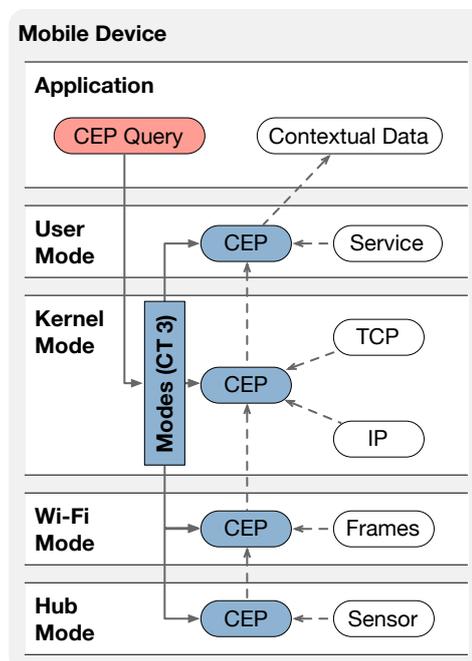
**Figure 6.2 ReactiFi: Reactive programming of Wi-Fi firmware (ReactiFi)** in transitional near-\* computing. Existing architectures already integrate different network- and application services for mobile users close to the network’s edge (orange). Complementarily, ReactiFi allows programs to be performed near-data and near-network, either for providing a service (blue), or for controlling transitions between wireless network protocols (light blue).

## 6.2 Multimodal Complex Event Processing

In this chapter, the novel concept of multimodal Complex Event Processing (CEP) is introduced. First, its relation to transitional computing is described in Section 6.2.1. The design and implementation of multimodal CEP is outlined in Section 6.2.2 and in Section 6.2.3. Section 6.2.4 presents experimental results. Section 6.2.5 discusses related work. Finally, this chapter is summarized in Section 6.2.6.

*Parts of this section have been published in [Gra+18b].*

### 6.2.1 Computational Transitions



**Figure 6.2.1** Computational Transitions within multimodal CEP.

Figure 6.2.1 depicts computational transitions within the approach of multimodal CEP. At the top, filtering, aggregating, and correlating operators on event streams are expressed by users or by applications in a high-level language that allows developers without domain knowledge to formulate complex queries. Multimodal CEP enables developers to efficiently detect user activities, gather environmental conditions, or analyze computing service, operating system, and network events. Multimodal CEP then automatically breaks up CEP queries and selects the most suitable execution mode for the involved CEP operators (CT 3). This allows irrelevant components of a mobile device to be turned off or put into sleep mode, and avoids unnecessary CPU cycles, such as context switches and memory copy operations. Event streams are processed on-device 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). Filter, aggregation, and correlation operators can use different event sources: hardware events from sensors, operating system events like network events from MAC, IP and Transport layer, and higher-level events from other services.

**Table 6.2.1** Operator algebra for event streams [KS09].

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

## 6.2.2 Design

In this section, the operator algebra (Section 6.2.2.1), the modes of multimodal CEP (Section 6.2.2.2), the query language (Section 6.2.2.3), and the mode selection algorithm (Section 6.2.2.4) are presented.

### 6.2.2.1 CEP Operator Algebra

We use a common operator algebra for event streams to define CEP operators [KS09]. 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 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 ( $1 < m \leq \text{size}$  time units or events) fashion. In addition to the window operator, the algebra consists of three operators, summarized in Table 6.2.1. 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).

### 6.2.2.2 Modes

Basically, 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 context [Bel+12]: user activities (user processes), system events (operating system), network events (Wi-Fi chip), and physical sensor values (sensor hub). Figure 6.2.2 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 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 co-processor

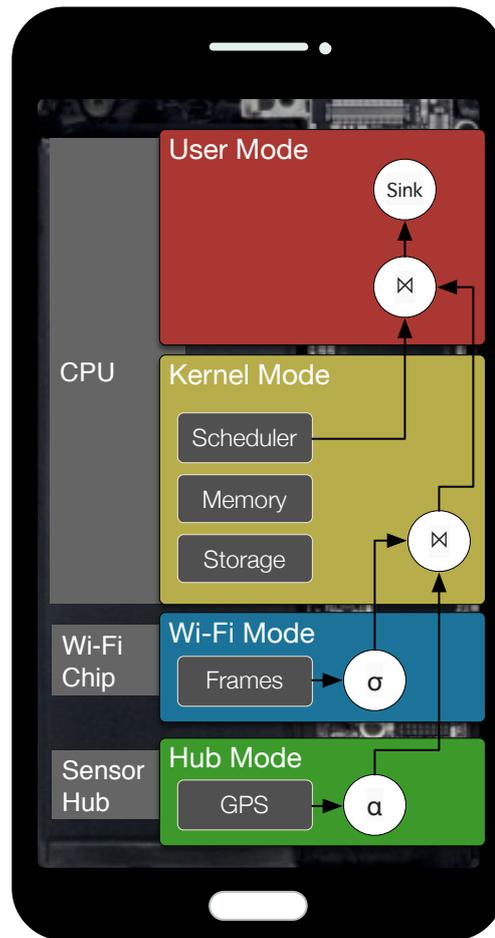


Figure 6.2.2 CEP operators in different modes.

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<sup>1</sup>. A kprobe 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)<sup>2</sup>, 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<sup>3</sup> 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

<sup>1</sup><http://www.kernel.org/doc/Documentation/kprobes.txt>

<sup>2</sup><http://www.kernel.org/doc/Documentation/networking/filter.txt>

<sup>3</sup><http://nexmon.org>

pressure and a proximity sensor, and a GPS sensor with an 8-bit microcontroller.

### 6.2.2.3 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. [ABW06].

**Listing 6.1** CQL query example

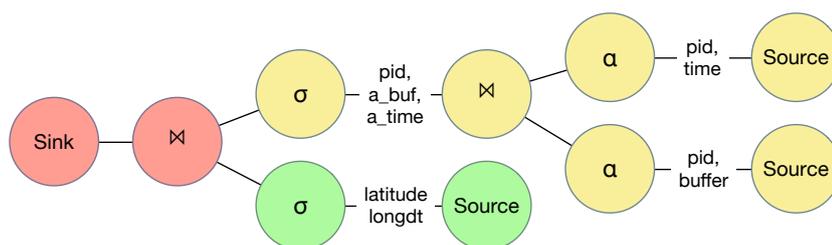
```
SELECT AVG(len) AS len FROM
  (SELECT * FROM kernel.sys_write WHERE pid==0)
  WINDOW(COUNT 300 JUMP 300)
```

Listing 6.1 shows 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 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 6.2** Quality-of-Experience (QoE) query example

```
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 6.2 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. In more detail, the kernel function `tcp_cleanup_rbuf()` of the TCP stack as a first input stream from kernel mode cleans up the receive buffer for full frames taken by the user, before an acknowledgment is sent. It contains the number of bytes the `tcp_recvmmsg()` 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 6.2.3** Quality-of-Experience (QoE) query example.

Figure 6.2.3 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 closely as possible to the event source, so 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.

#### 6.2.2.4 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 so 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 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. After that, 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<sup>4</sup>. 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

<sup>4</sup>Sat4J, [www.sat4j.org](http://www.sat4j.org)

Operator		$F_{in}$	$F_{out}$
Filter	$\sigma_\varphi$	$S[0].F_{out}$	$F_{in} \cdot sel(\varphi)$
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 6.2.2** Calculation of output event rate based on defined windows and predicate selectivity ( $sel$ ).

$C_{cpu}$  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 6.2.2 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  $\mathcal{O} := \{O_1, \dots, O_n\}$ , we define the set of decision variables  $X := \{x_1, \dots, x_n\}$  as:

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

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:

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

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:

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

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

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  $\mathcal{P}_{\mathcal{O}} := \{P_1, P_2, \dots\}$  as the set of all longest paths for the operator candidate set  $\mathcal{O}$ . A path is a sequence of operators  $P_i := \langle O_{i,1}, \dots, O_{i,n} \rangle$ , such that all operators are connected:  $\forall j \in \{1, \dots, n-1\} : O_{i,j} \in O_{i,j+1}.S$  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:

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

Here,  $Cap_{CPU}$  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) in the initial problem definition.

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 userland (i.e., not placed in kernel mode), all the operators should be directly processed in userland 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 evaluating 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:

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

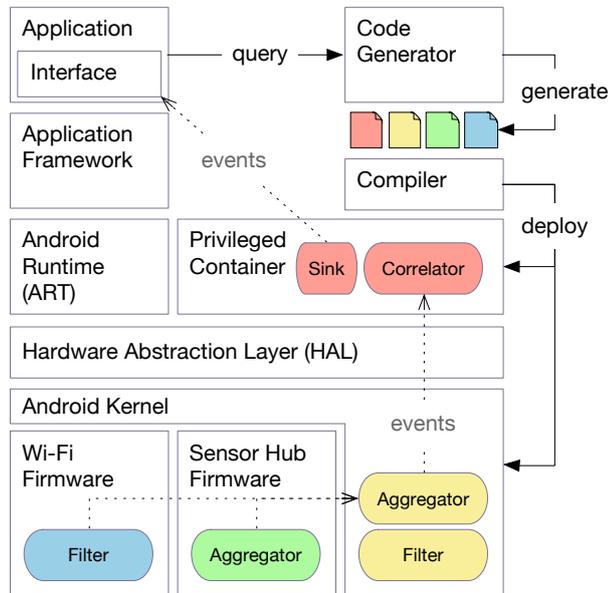
Here,  $\overline{\mathcal{O}}$  is the set of artificial operators and  $D(\mathcal{O}, src)$  extracts all operators of the candidate set  $\mathcal{O}$  that are directly connected to source  $src$ . These constraints act as a mutex allowing to either place the artificial operator (i.e., no operator at all) 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.

### 6.2.3 Implementation

In this section, the architecture of our multimodal CEP engine (Section 6.2.3.1), the assembly of mode-independent CEP operators to compilable mode-specific programs (Section 6.2.3.2), their execution model (Section 6.2.3.3), and our mode-specific implementation (Section 6.2.3.4) are presented.

#### 6.2.3.1 Architecture



**Figure 6.2.4** Multimodal CEP architecture in Android.

Figure 6.2.4 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 with device

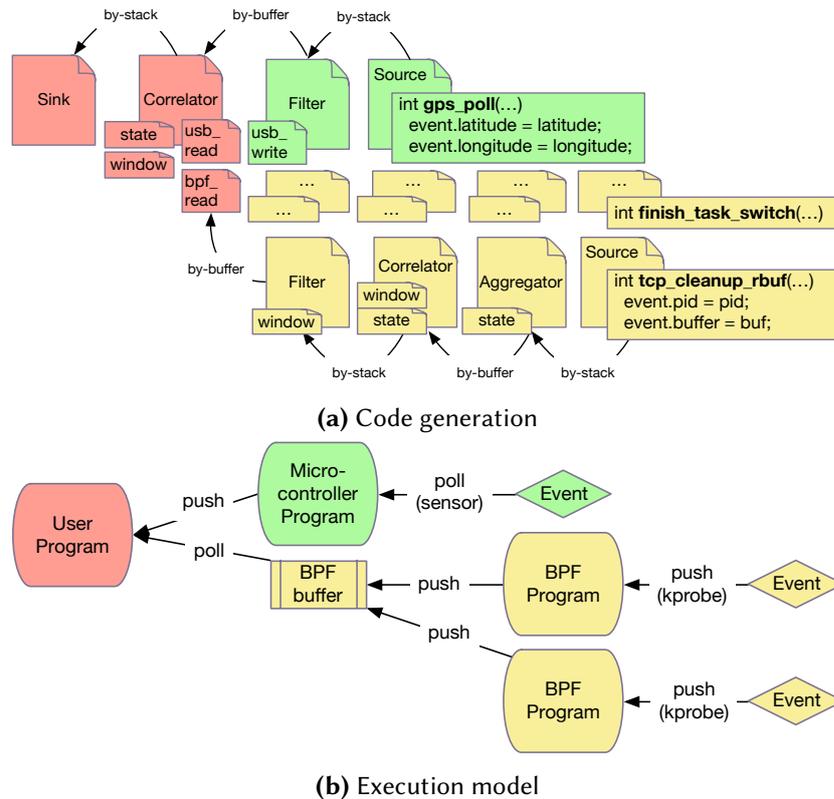
drivers, the Hardware Abstraction Layer (HAL) with vendor-specific device implementations, the managed Android Runtime (ART) used by applications, and the Application Framework for high-level services, such as activities and notifications. Applications formulate CEP queries for the multimodal CEP engine. 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 6.2.3.2, the compiler translates the generated code into mode-specific programs and deploys the programs according to Section 6.2.3.3. 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 privileged container, not as part of applications inside the ART. Our experiments have indicated 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, more importantly, 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 for high event rates.

### 6.2.3.2 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 – as a result, 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 (hub and Wi-Fi mode) and executable programs (user and kernel mode).

Figure 6.2.5a 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 6.2.5a, our code generation component weaves a producer function into the event



**Figure 6.2.5** Execution model and code generation for the Quality-of-Experience (QoE) query example.

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 by using annotated data structures in the producer functions. First, input streams are defined by producer function names, in case of Listing 6.2: `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.

### 6.2.3.3 Execution Model

Figure 6.2.5b shows the execution model of multimodal CEP using the Quality-of-Experience (QoE) example in Listing 6.2. 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 be 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.

### 6.2.3.4 Modes

In the following, we present implementations of kernel mode, Wi-Fi mode, and hub mode. They are based on the generic C implementation discussed in this 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<sup>5</sup>, 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 userland 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.

We use the BPF Compiler Collection (BCC)<sup>6</sup> 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.

<sup>5</sup><https://www.linaro.org/blog/kprobes-event-tracing-armv8>

<sup>6</sup><https://github.com/iovisor/bcc>

The BCC consists of an LLVM clang frontend<sup>7</sup> 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 userland. 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 userland program loading the bytecode and calling the `bpf()` system call. The userland 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 userland program via a file descriptor.

### Wi-Fi Mode

Wi-Fi chips of today's smartphones cannot be programmed. Consequently, our implementation of the Wi-Fi mode is based on the Nexmon firmware patching framework<sup>8</sup> 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 *ioctl*s.

In theory, the BCM4339 chip includes a total of 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 the 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_buff`s that are typically about 2 kB in size. Thus, queries can store about 50 `sk_buff`s. 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, co-processors 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 co-processor. Although most smartphone manufacturers do not specify which hardware is used for such co-processors, it is known, for example, that Samsung's Galaxy S4 uses a 32-bit Atmel AVR UC3 microcontroller<sup>9</sup> with 128 kB of flash memory, an operating

---

<sup>7</sup><https://clang.llvm.org>

<sup>8</sup><https://nexmon.org>

<sup>9</sup><http://www.techinsights.com/teardown.com/samsung-galaxy-s4>

frequency of 50 MHz, and 32 kB RAM<sup>10</sup>. Typically, current sensor hubs process events from, e.g., accelerometer, gyroscope, and compass sensors.

Since sensor hubs are usually closed-source, we 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 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 used the following sensors: (a) the L3GD20H gyroscope, (b) the LSM303 accelerometer and compass, (c) the BMP180 MEMS pressure sensor, (d) the VCNL4010 proximity sensor, and (e) the MTK3339 GPS sensor. The sensors (a)-(c) 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-based memory management for sensor values. If an event is needed (i.e., at least one operator uses this event), the reference count is increased. As soon as 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 hence not at the runtime of the program, the microcontroller must be flashed each time a new operator 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.

## 6.2.4 Experimental Evaluation

In this section, we present three evaluations: a high-throughput benchmark comparing kernel and user mode (Section 6.2.4.1), a network-event benchmark comparing Wi-Fi and kernel mode (Section 6.2.4.2), and a sensor hub evaluation (Section 6.2.4.3).

### 6.2.4.1 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

<sup>10</sup><http://www.atmel.com/devices/ATUC128L4U.aspx?tab=parameters>

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. We define an event as a one-byte write to an in-memory character device (`/dev/null`), which basically is a single system call and an in-memory write. This is similar to benchmarking a CPU with in-memory copies. Compared to benchmarks using other kernel subsystems (e.g., sending packets via the loopback device), the CPU spends 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<sup>11</sup>. For a branching instruction, the verifier has to check both the true and the 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.

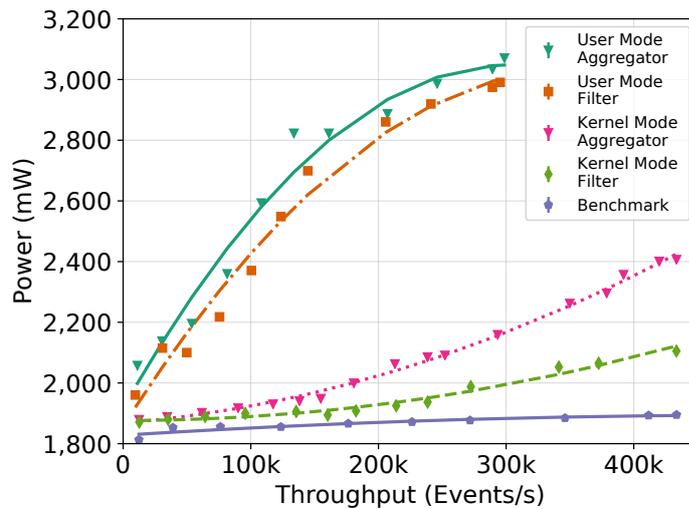
Figure 6.2.6 shows the results. The benchmark executed without any modifications is plotted as a baseline close to the bottom. Both the aggregator and the correlator were executed with a window size of 10 elements, no matter if they run in user mode or kernel mode. The aggregator predicate consists of 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 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 userland. 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 much 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.

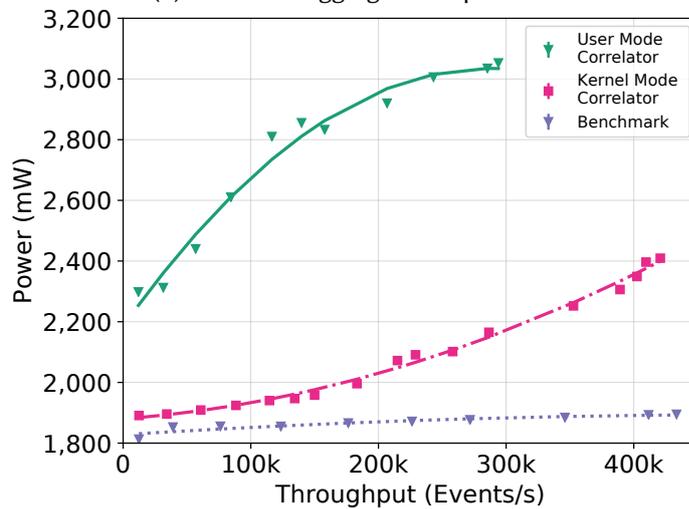
In addition, the maximum throughput for correlators and aggregators in user mode (about 300k events/s) is approximately 30% smaller than in kernel mode (about 430k events/s). Filters can process about 600k events/s, consuming about 2,400 mW power. The benchmark results also show that the maximum throughput for both user mode and kernel mode is only a fraction

---

<sup>11</sup><https://github.com/torvalds/linux/commit/17a5267>



(a) Filter and Aggregation Operators.

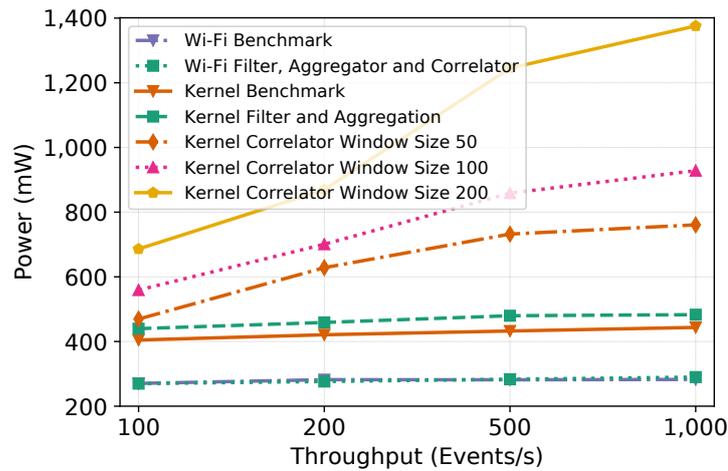


(b) Correlation Operator.

**Figure 6.2.6** Power consumption of CEP operators in user mode and kernel mode while processing system events.

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



**Figure 6.2.7** Power consumption of CEP operators in Wi-Fi and kernel mode.

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 userland implementation. Assuming a constant rate at 10k events/s, this will lead to a 7.5%, 12%, 21% longer battery lifetime.

#### 6.2.4.2 Wi-Fi Mode

We performed several experiments based on Nexus 5 smartphones (Broadcom BCM4339 Full-MAC 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  $\mu$ A. 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 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 6.2.7 shows the experimental results. The power consumption of operators in Wi-Fi mode is 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 result from the problem that the Wi-Fi chip is limited in its computational capabilities, as indicated by our correlation tests with larger window sizes.

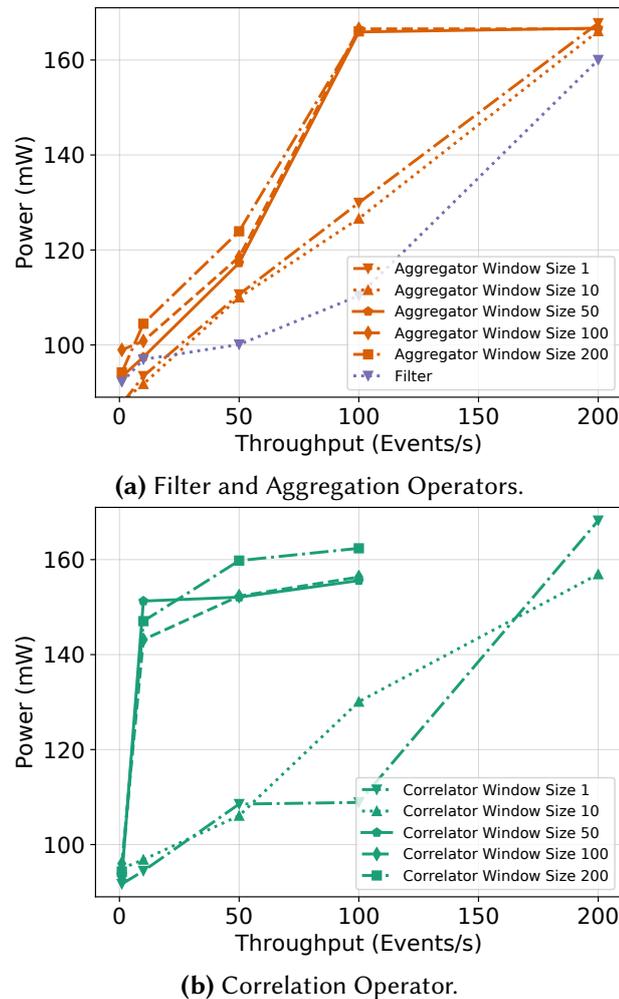
### 6.2.4.3 Hub Mode

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<sup>2</sup>C 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 6.2.8 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). By 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.



**Figure 6.2.8** Power consumption of CEP operators on the ATmega328 microcontroller.

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.

#### 6.2.4.4 Use Case: Mobility Data

In this section, we present a use case for gathering and analyzing mobility data by Wi-Fi probe request tracking with multimodal CEP. Probe request tracking [Fre15] is used for social link detection [CKB14] and geolocation tracking [Van+16] to gather mobility data sets, to collect mobility information about customers, and to monitor people in stores and shopping centers. Wi-Fi probes are part of the Wi-Fi 802.11 standard. They can be used for active service discovery by mobile devices through periodically broadcasted Wi-Fi frames that are collected by nearby stationary Wi-Fi access points and then sent to a central server for further analysis. In contrast, in our approach Wi-Fi frames are monitored on-device with multimodal CEP to create a mapping between the location of a device and the number of nearby Wi-Fi probe senders, e.g., to support crowdsensing applications.

**Listing 6.3** Correlating probe packets and GPS positions

```

SELECT * FROM
  (SELECT src, COUNT(*) FROM
    (SELECT * FROM wifi.frames WHERE type=0x00 AND subtype=0x04)
    WINDOW(COUNT 500 JUMP 500)
  GROUP BY src)
WINDOW(TIME 1 S JUMP 1S),
GPS@(1 Hz) WINDOW(TIME 1 S JUMP 1S);

```

**Listing 6.4** Outdoor detection query

```

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);

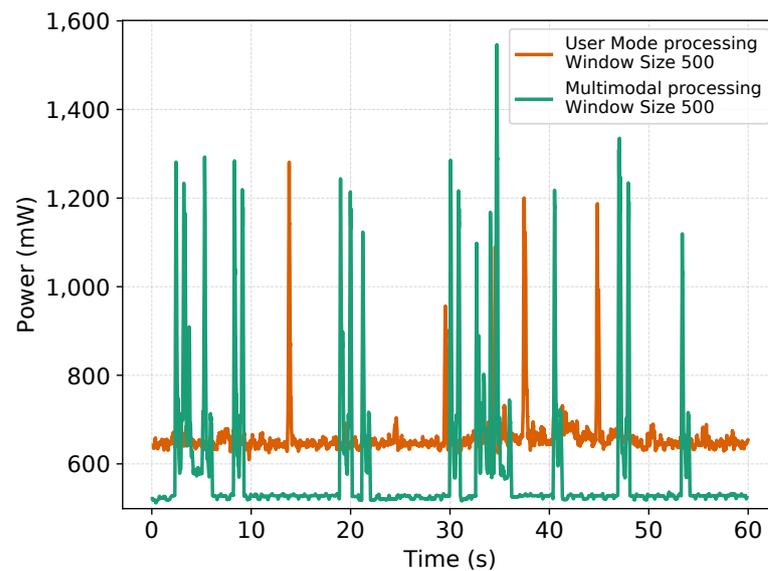
```

Listing 6.3 shows our multimodal CEP query. First, it consists of a correlation between a filter on Wi-Fi frames, which are grouped by their source address and counted. It has a window size of 500 elements, i.e., it is designed to deliver values after having received 500 probe packets. 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 probe request frame counts.

Since GPS is unreliable and power-intensive indoors, the query should only be executed outdoors. This is achieved by an outdoor detection query shown in Listing 6.4. It consists of two filters, where 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.

We performed experiments with two different implementations: (i) with an implementation where all frames and sensor values are pumped into a user program, (ii) with multimodal CEP. With multimodal CEP, the outdoor detection query and the GPS part of Listing 6.3 were executed on the sensor hub, while only the result of the correlation was sent to the process in user mode. The Wi-Fi probe request frame filter and the aggregation were executed on the Wi-Fi chip. The final correlator operated in user mode. In the other implementation, all queries were executed in user mode, while both sensor hub and Wi-Fi chip sent all data to the process in user mode. Additionally, to show the impact of the network transfer to a remote server, both implementations periodically sent the result of the queries to a server using a HTTP POST method.

Figure 6.2.9 shows the power consumption of the Nexus 5 smartphone and the sensor hub. It indicates a groundtruth of about 660 mW. As soon as 500 Wi-Fi probe request frames are counted, the upload requires more power (i.e., up to 1.3 W). If the queries are executed by multimodal CEP, the test hardware only consumes about 564 mW while executing the query on the sensor hub and the Wi-Fi chip, respectively. The upload, on the other hand, also requires about 1.3 W, in rare cases about 1.6 W. Additionally, during the multimodal tests, the power consumption shows more peaks, due to the more complex communication between Wi-Fi chip, sensor hub,



**Figure 6.2.9** Power consumption of a Nexus 5 processing Wi-Fi probes w/ and w/o multimodal CEP.

and user mode. In contrast to the user mode tests, the main CPU has to be woken up and put back into sleep in the multimodal tests, which results in additional power consumption.

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%. Since the Nexus 5 has a battery with a capacity of about 9,660 mWh, queries in our multimodal approach could be executed continuously for about 17 hours, whereas the battery would only last about 14 hours if queries were executed in user mode.

### 6.2.5 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  $\mu$ CEP engine [Akb+15] features dynamic rule updates in IoT devices. For embedded wireless devices, operating systems like TinyOS [Lev+05] and a corresponding query processor TinyDB [Mad+05] 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 [Mil+08] to dynamic/adaptive [Kjæ+09] duty-cycling of sensors up to sophisticated processing pipelines for various sensors [Lu+10]. Existing federated CEP systems [Bot+09; PH15] 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. [Mar+16] 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 [Ant+14] 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 of 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. So, multimodal CEP focuses on filtering, aggregating and correlating events within the existing mobile hardware/software architecture (mode).

Furthermore, multimodal CEP is related to other approaches to increase the energy efficiency of mobile devices. Lentz et al. [LLB15] optimize short-lived events in Android systems, where on->suspend and suspend->on transitions dominate the energy consumption. The authors construct a dependency graph between woke-up threads and other threads / components. They do not wake up all threads / components (unlike Android does), but a minimal set of dependent threads / components. All processes still rely on the main CPU to be executed, which however 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 [Lia+16; She+15] 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 [CC12], virtual network functions [AAS16], performance monitoring, dynamic tracing and security monitoring [HLL16]. For example, Cheng et al. [CC12] 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, which 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.

### 6.2.6 Summary

In this chapter, multimodal CEP was presented, a novel approach to process streams of events on-device 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), providing significant improvements in terms of power consumption and throughput.

Multimodal CEP realizes transitions between modes of CEP operators (CT 3). It automatically breaks up CEP queries and selects the most adequate execution mode for the involved CEP operators. Filter, aggregation, and correlation operators can be expressed in a high-level language without requiring system-level domain-specific knowledge.

We presented 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

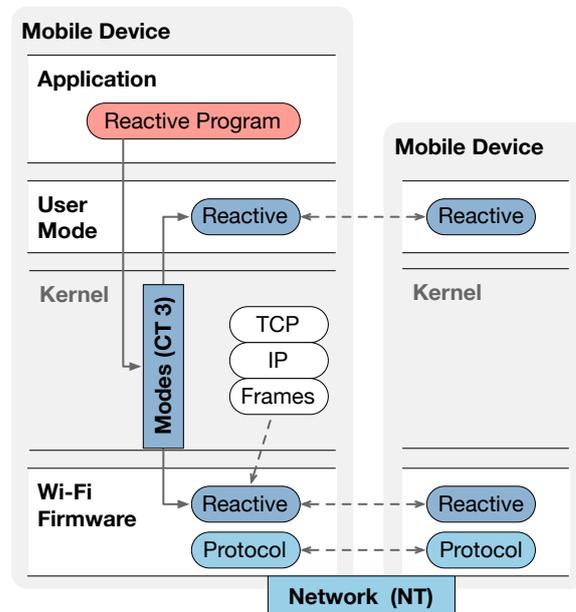
mode (kernel mode) compared to user mode, since more components of the SoC can be set to sleep mode and unnecessary CPU instructions are avoided. Up to 60% power can be saved if operators are executed on the Wi-Fi chip instead of in the kernel. We measured throughput improvements of up to 52% in kernel mode compared to user mode. The feasibility of our approach was demonstrated by a use case for gathering and analyzing mobility data by Wi-Fi probe request tracking.

## 6.3 Reactive Programming of Wi-Fi Firmware on Mobile Devices

In this chapter, the novel concept of reactive programming of Wi-Fi firmware (ReactiFi) on mobile devices is introduced. First, its relation to transitional computing is described in Section 6.3.1. The design of the ReactiFi domain-specific language and the runtime environment are presented in Section 6.3.2. In Section 6.3.3, ReactiFi implementation issues are discussed. An experimental evaluation, including two use cases to demonstrate the benefits of ReactiFi, are presented in Section 6.3.4. Section 6.3.5 discusses related work. Finally, this chapter is summarized in Section 6.3.6.

*Parts of this section have been published in [Ste+].*

### 6.3.1 Computational Transitions



**Figure 6.3.1** Computational Transitions within reactive programming of Wi-Fi firmwares on mobile devices.

Figure 6.3.1 depicts computational transitions within the approach of reactive programming of Wi-Fi firmware (ReactiFi). At the top, application programmers develop reactive programs in a high-level language to operate on events at different layers of the networking stack. ReactiFi programs are able to respond to PHY- and MAC layer events without involving the main CPU, that improves power consumption and latency considerably. Reactive programs consists of a dynamic graph with reactivities as nodes and dataflow dependencies as edges. Transitions are performed between the modes of a reactive (CT 3), as well as between MAC and IP layer network mechanisms (NT).

### 6.3.2 Design

In general, reactive programming allows a programmer to process asynchronous data streams. A reactive application consists of a dynamic graph with *reactives* as nodes and edges representing dataflow dependencies between them. Thus, reactive programming is particularly suitable to process network data streams.

Typically, reactive programs are written in a high-level language. We designed our ReactiFi DSL using a subset of the TypeScript language (version 2.7) for programming *reactives*. TypeScript is a superset of JavaScript with optional static typing and support for event-driven, imperative and functional programming. Our novel syntactical subset of Typescript enforces static typing and restricts the dynamic modification of objects' methods and functions, such that reactives can be compiled to executable binaries for the target platform.

Listing 6.5 shows a simple example of a reactive program with a stream of raw Wi-Fi frames captured in monitor mode.

**Listing 6.5** Reactive program example.

```
1 | var ethernet = RawStream
2 |   .map((h: ethernetHeader) => h.type)
3 |   .buffer(10)
4 |   .broadcastSend();
```

Reactives are defined as methods on observable objects. In this example, the starting point is the built-in `RawStream` that represents the stream of received frames. The `map` operation creates an observable object. A `buffer` operation that buffers elements (10 elements in the example) subscribes to the created observable object. The `broadcastSend` method that broadcasts the received event in the network subscribes to the created observable object in the same way. While the `RawStream` observable is a built-in object, the three operations are defined in a TypeScript library not shown here. To summarize, this program defines three reactives with data dependencies between them. It extracts the layer 2 header type field from each Wi-Fi frame and broadcasts a custom frame containing the last 10 layer 2 frame types received.

#### 6.3.2.1 ReactiFi Applications

There is a wide range of applications that could benefit from the programmability of Wi-Fi chips as supported by ReactiFi. For instance, advanced scheduling algorithms for wireless networks often consider PHY-related parameters, such as rate adaptation, time-varying channels, and heterogeneous delay bounds [Hou14]. They depend on the characterization of flows in terms of traffic patterns, channel reliability, delays, and throughput [HK09] in realtime. Hence, implementations of such scheduling algorithms might profit from ReactiFi by processing PHY-related features directly on the Wi-Fi chip.

Similarly, dynamic and cooperative sharing of spectrum resources can be achieved by advanced medium access control (MAC) with database assisted and dynamic spectrum access [Che+16b; DGH14], which could be supported by programmable Wi-Fi chip firmware with low power consumption.

Das et al. [Das+17] use Wi-Fi congestion information to improve the quality adaptation algorithm used in Skype. Wi-Fi congestion is calculated using the QoS extension introduced in 802.11e, i.e., four sending queues with different priorities. These priorities are directly mapped to the Type of Service header field of the IPv4 protocol. Sending an ICMP request with low priority and another one with high priority is used to estimate congestion. By counting the packets grouped by applications, it can be estimated whether congestion is caused by full Wi-Fi queues or other reasons. ReactiFi can be used to react to Wi-Fi congestion as proposed by Das et al.

MoFA presented by Byeon et al. [Bye+14] adapts Wi-Fi frame aggregation based on user mobility, which is estimated using the subframe error rate, i.e., the number of failed subframe transmissions. If the estimated mobility is higher than a threshold, the size of aggregated frames will be reduced, which increases throughput in mobile environments. If the environment is categorized as static, the size of aggregated frames will be increased beyond the default settings. ReactiFi can be used to adapt Wi-Fi frame aggregation in an efficient manner.

Sun et al. [SSK14] show that client roaming, rate control, and beamforming can be improved using mobility information derived from the PHY layer. The mobility of a user is calculated using channel state information (CSI) and Time-of-Flight (ToF). With CSI, an AP can identify and distinguish the multiple paths of an arriving Wi-Fi signal. If a frame is sufficiently dissimilar compared to its predecessor, it is assumed to have arrived from another path, indicating a mobile user. To confirm the CSI check, ToF is used to determine whether the ToF of two consecutive frames in a static environment is similar. Although this approach is implemented on the AP and not on the client, ReactiFi could help to use the same information for improvements on the client.

ReactiFi can also be used to support data acquisition and in-network data processing applications with collaborative users [Dua+12]. Chen et al. [Che+17] propose multi-layered networks as fusions of networks from different domains, and formulate a cross-layer dependency inference problem as a collective collaborative filtering problem that may benefit from the efficient data processing capabilities offered by ReactiFi.

Li et al. [Li+17] present a crowdsourcing approach for network quality measurement that aggregates quality measurements from large numbers of users. This approach suffers from the fact that user-provided network quality measurements are affected by user-related factors, which may be a tradeoff between hardware capabilities, user mobility, and user habits. The analysis of user-related factors can be performed on end devices in a distributed fashion using ReactiFi.

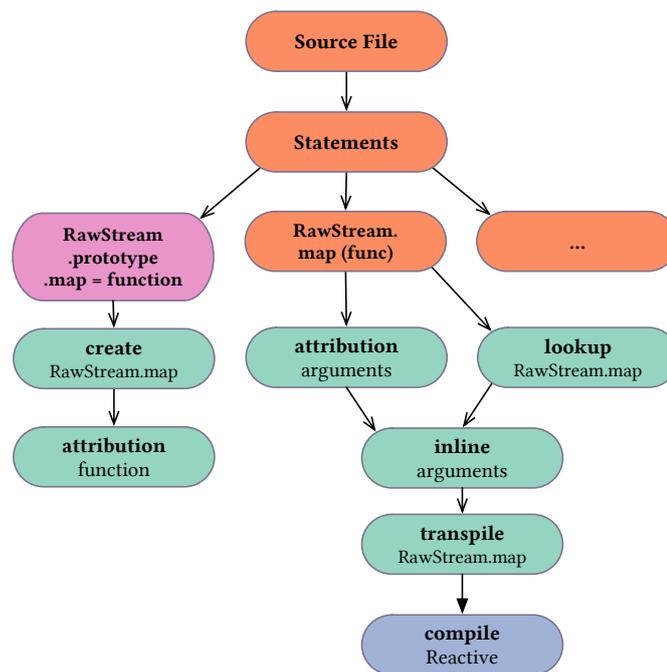


Figure 6.3.2 Compiling the reactive program in Listing 6.5.

### 6.3.3 Implementation

In this section, we discuss the implementation of the ReactiFi DSL and the corresponding runtime executed on the Wi-Fi chip.

#### 6.3.3.1 ReactiFi DSL Implementation

The compilation process of a program written in the ReactiFi DSL works as follows. The TypeScript abstract syntax tree (AST) is traversed to attribute its AST nodes with additional information for the transpilation (i.e., source-to-source translation) and compilation (i.e., source-to-binary translation) processes. The TypeScript syntax is first transpiled to C modules that are then compiled to executable binary blobs for the target platform. Each `Source File` and `Block` node declares a new scope for binding variable and function names to an accessor object. Accessor objects are the subject of the attribution process and represent the memory location of variables and functions as well as their static types, which are later used for the transpilation and compilation processes.

Figure 6.3.2 shows the compilation process for the reactive program shown in Listing 6.5. Pink elements are AST nodes referring to built-in functions provided by ReactiFi (for the first time available in a reactive language for developers at the application level), green elements represent actions during AST traversal, and the blue element represents a compiled module. Orange elements denote unmodified AST nodes of the TypeScript language. The source file at the top of the figure contains a set of statements. The first statement (left) declares a new member of the built-in type `RawStream`, for which a variable within the scope of the object `RawStream` is created. At its right-hand side, a `FunctionDeclaration` that contains argument definitions, a function body, and a return type are shown. In the second statement, a `CallExpression`

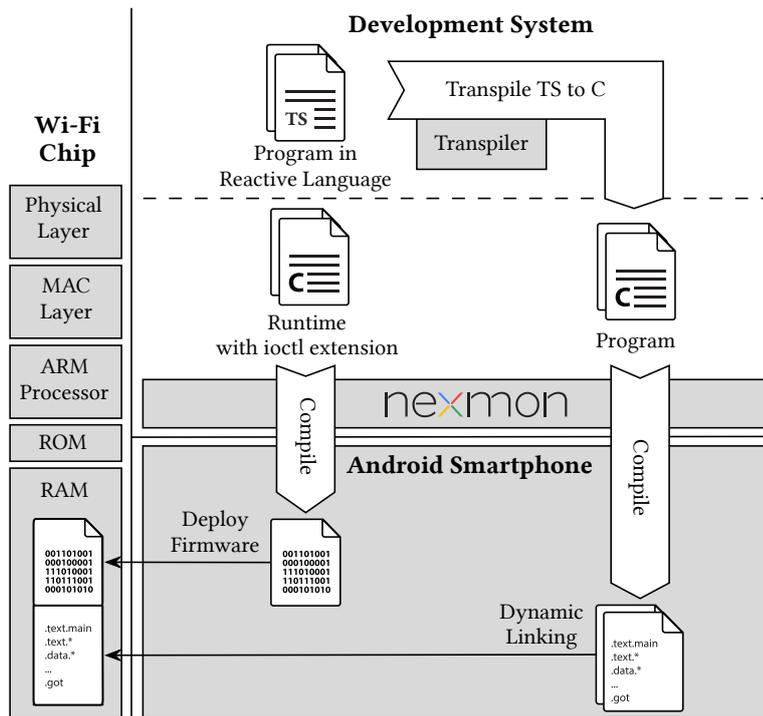


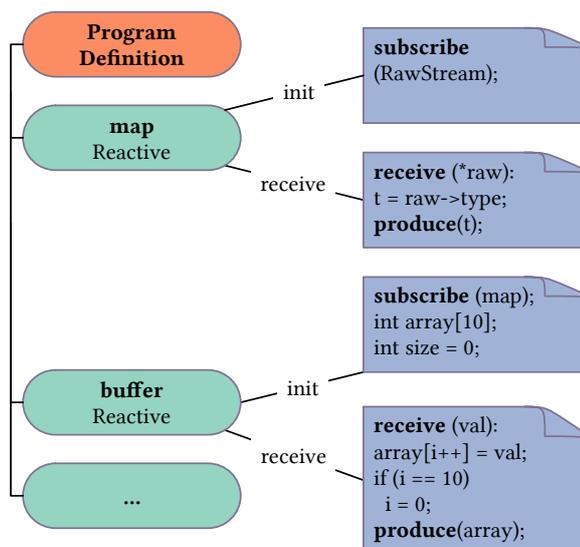
Figure 6.3.3 Extensions of Nexmon.

is shown, with a `PropertyAccessExpression` as the callee and a first-class function as the argument. The callee is looked up from the `RawStream` scope and used as input for the argument inlining action, which basically transpiles first-class functions to C statements. The output of the second statement is a compiled reactive. Using the ReactiFi DSL and the supplied tools, developers can develop reactive programs for Wi-Fi chips for the first time.

### 6.3.3.2 Patching the Firmware at Runtime

Manufacturers of Wi-Fi chipsets for commodity mobile devices distribute their firmware as closed source software. Developers of system software cannot modify the firmware, but have to rely on firmware interfaces to implement OS modules, such as Wi-Fi drivers that are executed on the main CPU. To be able to change the Wi-Fi firmware on commodity mobile devices, the binary firmware needs to be reverse engineered, patched, and compiled, unless manufacturers provide open source versions of their software. The developers of the Nexmon patching framework reverse engineered the firmware of several Broadcom Wi-Fi chipsets that have been integrated into commodity mobile phones like Nexus 5 or Samsung Galaxy S7. Nexmon allows system programmers to write patches in the C programming language, and automatically integrating them into the firmware.

Position independent code (PIC) modules are C modules compiled to binary blobs that can be transferred to the firmware and be loaded to arbitrary memory addresses from where their execution can be triggered during runtime. To provide this functionality, we developed the Nexmon PIC extension, for the first time enabling PIC on a Wi-Fi chip for developers. These files all contain a `main` function that we always place directly at the beginning of the reloadable



**Figure 6.3.4** Reactive program definition.

file. To call the main function, we extended the firmware to branch into the recently loaded code. After the main function, we placed additional functions and variables, followed by a global offset table (GOT). The latter is required to make the code position independent. Since we do not know where the binary blob is loaded during runtime, the code within our blob needs to perform jumps relative to the program counter to reach code within this blob, while existing firmware functions need to be accessed by first loading the absolute target address from the GOT into a register and then jumping to the loaded address.

As shown in Figure 6.3.3, we created an ioctl to load an executable binary blob containing the program to be executed in the reactive runtime. The serialization module first serializes the binary that is loaded with the ioctl onto the heap of the Wi-Fi chip and stays there until the ioctl process has finished. Hence, we can directly execute the loaded code by creating a function pointer to its starting address and calling it. Since the binary blob should be executable outside the ioctl function, i.e., in a reactive that processes incoming frames, we first copy it into a newly allocated section in the heap so that it is not overwritten after the ioctl processing has finished. Then, we can freely pass its starting pointer to other functions to call the new code. Thus, our implementation supports the development and deployment of programs at any time without interrupting Wi-Fi connectivity.

### 6.3.3.3 Runtime Execution Model

Figure 6.3.4 shows a program definition that includes the definition of reactives. Within a program compiled for the target platform, every reactive definition contains an `init` and a `receive` routine. The `init` routine is used to initialize a reactive, i.e., to specify its initial subscriptions and to initialize local variables and reserve sufficient memory in the memory management module. The `receive` routine is the main routine for a reactive that is called when a new event is received. In this example, the `map` and `buffer` reactives were compiled from Listing 6.5. While the `map` reactive consists of a simple transformation, the `buffer` reactive fills an array of ten input events. The size of this array (in bytes) is inferred from the event type

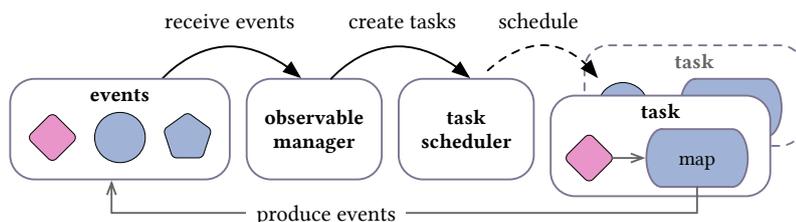


Figure 6.3.5 Scheduling reactives.

of its subscription, and built-in observables and arrays are internally handled as pointers to in-memory data structures using the memory management module. In case of the `RawStream`, it represents a pointer to captured Wi-Fi frames stored in main memory. The array in the scope of reactives of `buffer` represents a pointer to the local array. Single elements and attributes of these data structure can be accessed and their values can be copied, but these data structures are not duplicated and not stored for later use.

#### 6.3.3.4 Scheduler for Reactives

Due to real-time constraints for frame and packet handling on a Wi-Fi chip, executing long-running reactive programs is not possible. In `ReactiFi`, programs are split into reactives that are executed asynchronously with the help of a novel non-preemptive efficient task scheduler. Figure 6.3.5 illustrates its functionality. Events from built-in observables (pink) and user-defined reactives (blue) are received by an observable. Then, for each subscriber of this observable, a new asynchronous task is created, containing a reference to the subscribing reactive and the received event. Tasks created by the observable manager are enqueued in the run queue of the task scheduler and scheduled in a FIFO manner.

Furthermore, `ReactiFi` supports the execution of multiple programs. These are executed sequentially and can be loaded dynamically during runtime, without interrupting Wi-Fi connectivity, as described in Section 6.3.3.2.

Within a reactive program, built-in functions can be used to execute operations of the Wi-Fi firmware that have already been implemented and are ready to use, such as sending custom Wi-Fi frames (e.g., `broadcastSend` in Listing 6.5), sending packets to the network stack of the host OS, switching between 2.4 GHz and 5 GHz frequency bands, switching channels, (de-)activating monitor mode, or activating the 802.11z TDLS mechanism. These actions are triggered by our scheduler for reactives as soon as the task has finished its execution.

#### 6.3.3.5 Memory Management

The memory management module is responsible for storing linked programs, reactives, and the schedulers' run queue, using the concept of *pools*. A pool is a preallocated chunk of memory for objects of a certain size. It is comparable to a kernel slab allocator, but implemented as a ring buffer without override protection. Due to its fixed size, it ensures that the limited memory of the Wi-Fi chip is not exceeded. Moreover, the memory management module also passes parameters in a call-by-reference manner. Thus, a `sk_buff`, i.e., the C struct representing an entire Wi-Fi frame, or parts of it do not have to be held in multiple copies in the RAM of the

Wi-Fi chip, which helps to reduce the memory footprint of our implementation in a simple though effective manner.

To handle the limited amount of memory of typical Wi-Fi chips (e.g., the Wi-Fi chip of a Nexus 5 has about 100 kB usable RAM for ReactiFi programs), the required memory for every reactive program is inferred at compile time and allocated during initialization. We also check whether sufficient memory is available for the program to be executed to ensure that the Wi-Fi chip does not run out of memory, which would lead to crashes and non-functional Wi-Fi on the mobile phone.

The allocated memory is separated into distinct regions for each reactive and a global scope. While the global scope is accessible by all reactives, the reactive specific scopes can only be accessed by the reactive which they have been assigned to. Additionally, each program has its own memory pool separated from other programs.

### 6.3.4 Experimental Evaluation

In the following, the benefits of ReactiFi are demonstrated by two use cases. In Section 6.3.4.1, a nearby video streaming use case is presented, in which the Wi-Fi mode is switched dynamically between 802.11n and 802.11z TDLS. In Section 6.3.4.2, a virtual network stack using ReactiFi on the Wi-Fi chip is described as a second use case. It allows programmers to develop opportunistic networking applications while a mobile device is still connected to Wi-Fi APs. Use case (a) has been selected to demonstrate improvements in terms of throughput and quality of service. Use case (b) has been selected to demonstrate improvements in terms of power consumption and latency. Additionally, use case (b) indicates the limits of Wi-Fi programmability due to the limited computational power and memory capacity of the Wi-Fi chip.

#### 6.3.4.1 Use Case: Nearby Video Streaming

File sharing in local wireless networks is a common use case in consumer environments. For example, Apple offers “AirDrop”, Microsoft has “Nearby Sharing”, and Google provides the “Nearby” API for sharing files locally. Despite various available wireless transmission technologies, all of these products rely on Wi-Fi to transmit data. In a Wi-Fi environment, data can be transmitted either over an access point (AP) or directly between neighboring nodes. On the one hand, an AP covers a larger area, but can be congested or overloaded, especially in crowded scenarios like football or soccer games. On the other hand, a direct Wi-Fi connection to a neighboring node can achieve higher throughput, but only within an area that is quite limited.

Apart from file sharing, video streaming is an increasingly popular application. According to Cisco, IP video traffic will amount to 82% of all consumer Internet traffic by 2021 [Cis17]. Thus, it is quite likely that service providers will introduce technologies to extend nearby file sharing mechanisms to provide nearby video streaming. To demonstrate the rapid prototyping capabilities and potential performance improvements achievable by using ReactiFi, we present an illustrative nearby video streaming use case based on ReactiFi below.

**Listing 6.6** ReactiFi code for nearby video streaming.

```

var CNT: number = 50;

var myPacketCount = RawStream
  .filter((frame: ethernetFrame) => frame.dst == MY_MAC_ADDRESS)
5  .groupBy((frame: ethernetFrame) => frame.src)
  .count();

var avgPerSrc = RawStream
  .groupBy((frame: ethernetFrame) => frame.src)
10  .buffer(CNT)
  .reduce((snr: number, x: number) => snr + x)
  .map((snrSum: number) => snrSum / CNT);

var setupTdls = myPacketCount
15  .correlate(
    avgPerSrc, (no: number, average: number) =>
      no > CNT_THRESHOLD && average > SNR_THRESHOLD
  ).tdlsSetup();

20 var destroyTdls = myPacketCount
  .correlate(
    avgPerSrc, (no: number, average: number) =>
      no <= CNT_THRESHOLD && average <= SNR_THRESHOLD
  ).tdlsShutdown();

```

### ReactiFi Program

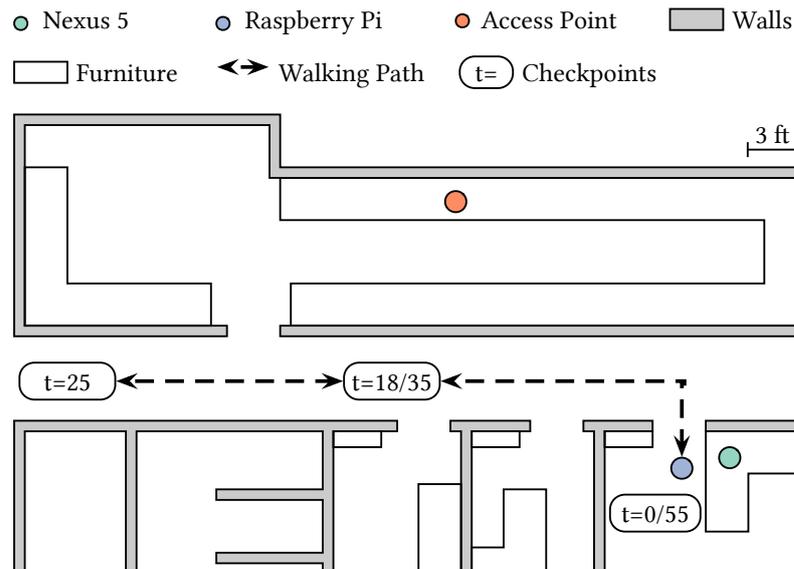
To get the best of AP based and direct wireless transmission approaches, we implemented a reactive program that inspects surrounding Wi-Fi signals. When a file or video is distributed in a local wireless network, the source sends the data to an AP that relays the same data to the destination, resulting in two data streams containing the same payload. If this behavior is detected by the video receiver using an ReactiFi program, it will switch from 802.11n AP mode to an 802.11z Tunneled Direct Link Setup (TDLS) connection. TDLS is used to establish a direct communication tunnel to another Wi-Fi device, without losing or disturbing the previously established connection to a wireless AP.

Listing 6.6 shows our implementation. First, we have to separate frames sent to the destination of the flow from all other traffic. Thus, the first reactive dissects the frame and inspects the frame control field in the Wi-Fi header. Depending on the configuration of the From DS and To DS bits, the inspected frame is either coming from an AP, going to an AP or is a frame sent via TDLS. For each of these three frame types, the four possible MAC addresses in the Wi-Fi header have to be handled differently. Based on the information in the frame control field, we can determine whether the frame is meant for the corresponding destination or not. If not, the frame is dropped by the Wi-Fi chip. This process is shown in a shortened version in lines 3 to 6 in Listing 6.6. Additionally, the average SNRs for all packets is computed. For every other node that communicates with the video receiver, a distinct average is calculated and stored, as shown in lines 8 to 12.

Finally, the stored averages are correlated as shown in lines 14 to 24: if the SNR to the video or file source is better than to the AP and currently TDLS is not set up, TDLS will be established. If

**Table 6.3.1** H.264 encoded video configurations for Dynamic Adaptive Streaming over HTTP (DASH).

Width	Height	Frame Rate	Bit Rate	Notation
1,280	720	30	3,000	720p
1,920	1,080	30	5,000	HD
2,880	1,620	30	7,500	3K
3,840	2,160	30	10,000	4K 30 FPS
3,840	2,160	60	20,000	4K 60 FPS

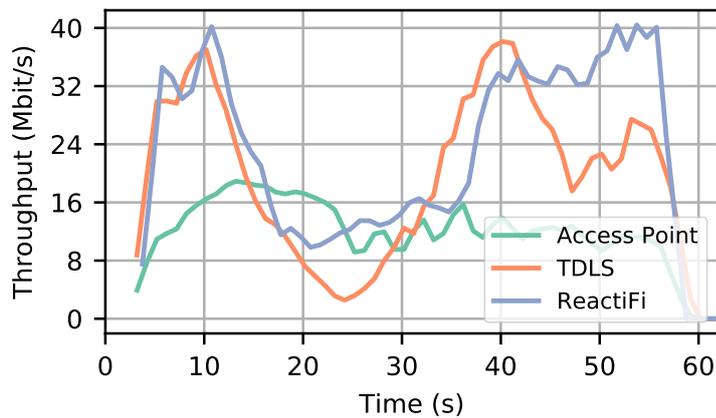
**Figure 6.3.6** Experimental setup and walking path with starting point ( $t = 0$ ), switch point from TDLS to AP mode ( $t = 18$ ), point with maximum distance ( $t = 25$ ), switch point from AP mode to TDLS ( $t = 35$ ) and end point ( $t = 55$ ).

the SNR to the AP is better than to the data source and a TDLS is already present, it will be dismantled, which is performed by the fourth reactive.

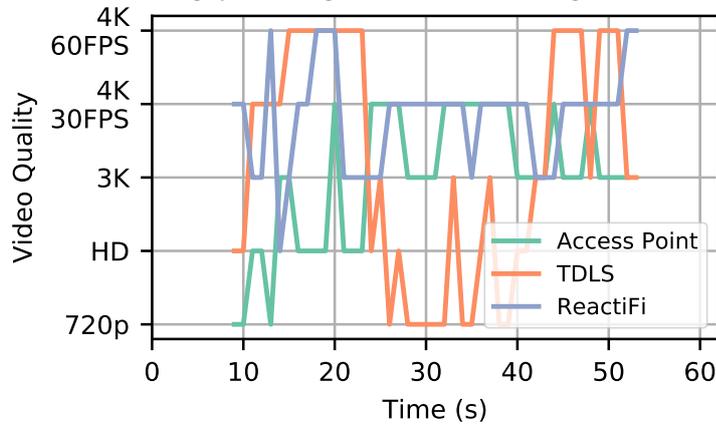
### Experimental Setup

To evaluate our implementation, we used a Nexus 5 smartphone as a client with the described reactive program installed. As a server, we used a Raspberry Pi 3 with the latest Nginx web server, which served a DASH video player and an H.264 encoded video with an adaptive bitrate. The video was encoded in five configurations up to 4K with 60 frames per second (FPS), as shown in Table 6.3.1.

Furthermore, we used a Turriz Omnia RTROM01 router with the OS version 3.2.1 in stock configuration as our AP. Wi-Fi was set to 802.11n mode on channel 6 in the 2.4 GHz band to increase the usable Wi-Fi range. Since the Nexus 5 was modified for power measurements (see Section 6.3.4.1) and thus not mobile anymore, we powered the Raspberry Pi with a battery pack. As illustrated in Figure 6.3.6, the AP was about 20 feet away from the stationary Nexus 5. We



(a) Throughput during three video streaming tests.



(b) Video quality during three video streaming tests.

**Figure 6.3.7** Throughput and video quality during three video streaming tests: using AP, TDLS, and ReactiFi.

started with the Raspberry Pi about 3 feet away from the Nexus 5 at  $t = 0$ , then we moved towards the AP up to 9 feet ( $t = 18$ ) and then farther away from both the AP and the Nexus 5. The maximum distance between Nexus 5 and Raspberry Pi was about 40 feet and 24 feet between Raspberry Pi and AP ( $t = 25$ ). After that, the same path was used for the way back ( $t = 35$ ), resulting in the same position as at the beginning of the test ( $t = 55$ ). The video was played for a total period of 55 seconds, while the maximum distance between Nexus 5 and Raspberry Pi was reached after about 25 seconds.

We logged both the SNR from Nexus 5 to AP and from Nexus 5 to Raspberry Pi. Additionally, we measured the power consumption of the Nexus 5 and logged video playback statistics of the DASH player, as proposed by Stohr et al. [Sto+17]. Finally, the outgoing bandwidth for the video playback was logged on the Raspberry Pi. Since all surrounding wireless traffic had to be analyzed for this application, the Nexus 5 was set to Wi-Fi monitor mode.

### Throughput and Video Quality

Figure 6.3.7a shows the throughput in Mbit/s on the y-axis and time in seconds on the x-axis during three tests. First, the video was streamed only via the AP, then only with TDLS, and finally ReactiFi with its described reactive program was used.

While the throughput in AP mode is almost constantly at about 12 Mbit/s, the same video served via TDLS shows peaks at about 40 Mbit/s at the beginning and end of the test, i.e., when the Raspberry Pi is very close to the Nexus 5 ( $t = 0$  and  $t = 55$ ). The fluctuations in the AP tests are caused by the DASH player, which only buffers a few seconds and then stops the download until the buffered content undercuts a certain threshold. At the maximum distance (i.e., the worst SNR) between the two devices ( $t = 25$ ), the throughput drops to 4 Mbit/s in the TDLS tests. The throughput in TDLS mode correlates with the distance between Raspberry Pi and Nexus 5, but the AP test seems to be agnostic to the distance.

The same experiment performed with the described reactive program shows significant improvements over both the TDLS and AP tests, as shown in Figure 6.3.7a. On the one hand, at the beginning and at the end (i.e., with the best SNR) of the test, the results are comparable to the pure TDLS test, where throughput exceeds 40 Mbit/s. But at the worst SNR, on the other hand, throughput does not undercut the values of the pure AP test.

These differences also can be seen in terms of video quality, as shown in Figure 6.3.7b. While it is possible to stream the video in 4K 60 FPS in TDLS mode over short distances, the video quality can drop to 720p, again correlating with the distance. The video quality via AP is again relatively constant at 3K, does rarely reach 4K 30 FPS and never 4K 60 FPS. In TDLS mode, video quality adapts 15 times, whereas in AP mode the quality changes only 12 times.

Using our reactive program shows significant improvement in terms of video quality. The video starts at 4K 30 FPS and even reaches 4K 60 FPS. Even at the worst SNR, the video quality stays above the results of the AP tests. The worst video quality measured during this test is HD.

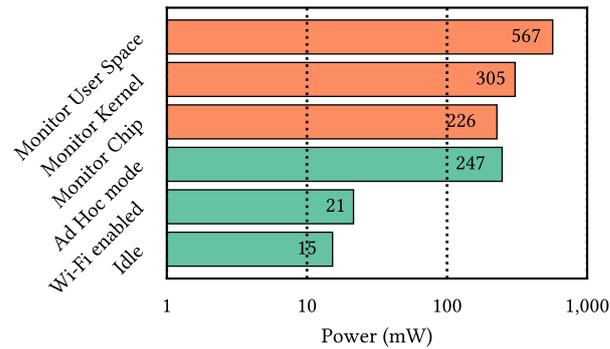
Furthermore, the number of quality adaptations is also less than in both other tests, with only 9 adaptations. These improvements are due to reactive switching of the two modes at  $t = 18$  and  $t = 35$ , respectively.

### Power Consumption

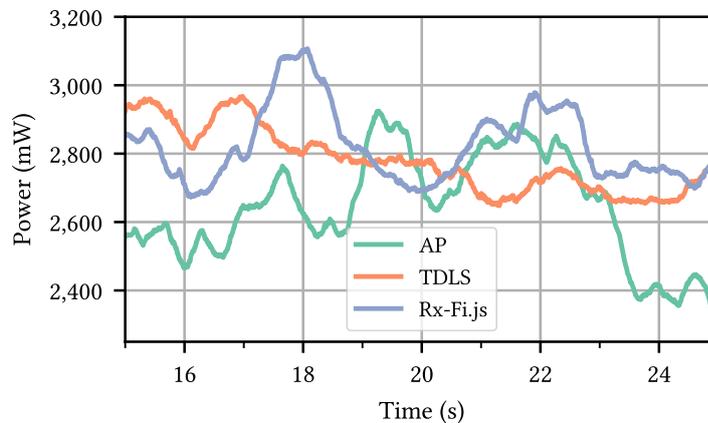
To evaluate the power consumption of our ReactiFi program, we first performed three idle tests and three tests with the Wi-Fi chip set to monitor mode, since our video streaming use case requires the Wi-Fi chip to be in monitor mode.

To perform power measurements without interferences from the battery as precisely as possible, 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 back into the Nexus 5. The measurements were performed using a Monsoon High Voltage Power Monitor with a sample rate of 5 kHz and a resolution of  $286 \mu\text{A}$ . The voltage was set to 4.2 Volts, which corresponds to about 92% battery capacity.

Figure 6.3.8 shows bars in two colors, representing idle (green) and monitor mode (orange) tests. The power consumption is shown on the x-axis on a logarithmic scale.



**Figure 6.3.8** Power consumption baselines: idle and monitor mode.



**Figure 6.3.9** Power consumption during video streaming via AP, TDLS, and ReactiFi.

The experimental results for idle mode are as follows. The lowest bar shows that the idle power consumption of the Nexus 5 is about 15 mW. With Wi-Fi enabled, power consumption increases by about 6 mW to approximately 21 mW, regardless of whether the Wi-Fi connection is associated with an AP or not. The idle power consumption of the Nexus 5 with Wi-Fi set to ad hoc mode is about 247 mW, as shown in the third bar.

The upper three bars show the power consumption in monitor mode. If all the captured frames are dropped in the Wi-Fi firmware and not passed to the main CPU, power consumption is about as high as in ad hoc mode. This is because ad hoc mode and monitor mode are technically similar, with the difference that in ad hoc mode the Wi-Fi firmware drops all packets not directed to the device. If all frames are transferred to the kernel and dropped there, power consumption increases by 80 mW. If the frames are passed to user space, the device consumes about 565 mW of power.

Finally, we used the benchmark app Antutu<sup>12</sup> to obtain the highest possible power consumption for this device, which can be up to 6 W (not shown in Figure 6.3.8).

The evaluation of the ReactiFi program shows that power consumption during these tests was dominated by video playback. While power consumption during the AP test was about 2.6 W, the TDLS test required 2.8 W of power, as shown in Figure 6.3.9. This overhead can be attributed

<sup>12</sup><http://www.antutu.com/en/index.htm>

to the following reasons. First, the monitor mode requires additional power, as already described above. Second, TDLS, which is similar to the monitor mode and technically comparable to the ad hoc mode, requires additional power. However, both effects are negligible compared to the power required for receiving more data and playing a video with higher bitrates.

During the tests with ReactiFi, power consumption was also about 2.8 W. Thus, even under heavy load, ReactiFi does not introduce power overhead compared to the TDLS test. When the video stopped playing at the end of the test, power consumption decreased to about 900 mW in our use case and the TDLS experiment and to 650 mW in the AP test, confirming our explanations about the power overhead.

These experiments show that our reactive Wi-Fi approach improves throughput and video quality significantly, leading to higher possible data rates and thus faster file transmissions. Moreover, our implementation only introduces negligible computational and power overhead.

### 6.3.4.2 Use Case: Virtual Network Stack

Using the Internet on mobile devices in crowded scenarios like soccer games or music festivals can be difficult if not impossible, due to overloaded cell towers and APs. Additionally, many locations of such events do not provide Wi-Fi, but rely entirely on cellular networks. Although cellular network providers sometimes deploy temporary cell towers, the network might still not be usable because of overload periods.

To remedy this situation, opportunistic networks can be used to enable local communication via delay-tolerant networking (DTN) and hop-to-hop routing to relieve the cellular infrastructure. However, the problem with DTN or hop-to-hop routing is that commodity mobile devices need either additional special hardware or software, which interferes with connections to APs, since it is not possible to be connected to an opportunistic network and a traditional network at the same time. If no Wi-Fi infrastructure is available due to simply missing APs or in disaster scenarios where electricity and communication infrastructures are destroyed or seriously damaged, ReactiFi can be used to provide power saving communication modes, as discussed below.

### ReactiFi Program

In particular, ReactiFi can be used to implement a virtual network stack on the Wi-Fi chip, as shown in Listing 6.7.

While the mobile device is connected to an AP, it is possible to be directly connected to other devices via TDLS and form a virtual network. If no APs are available, frames can be broadcast and received by other devices that are in monitor mode as a fallback solution. An application developer can use ICMP packets of type 255 (currently not used, thus free for ReactiFi). ReactiFi intercepts these packets on the Wi-Fi chip and distributes them in a hop-to-hop manner within local or DTN networks.

Based on this approach, several applications can be implemented. For example, event organizers can develop a mobile app that broadcasts notifications to all attendees in an epidemic manner. The notifications will be held in the Wi-Fi chip (line 11) until the user wakes up the device (not

**Listing 6.7** ReactiFi code for a virtual network stack.

```

var bufferArray = []

var outgoing = ICMPStream
  .filter((frame: ethernetFrame) => frame.type == 255)
5  .filter((frame: ethernetFrame) => frame.src == MY_MAC_ADDRESS)
  .sendFrame()

var imcommingMyPackets = ICMPStream
  .filter((frame: ethernetFrame) => frame.type == 255)
10  .filter((frame: ethernetFrame) => frame.dst == MY_MAC_ADDRESS)
  .append(bufferArray)

var imcommingRelay = ICMPStream
  .filter((frame: ethernetFrame) => frame.type == 255)
15  .filter((frame: ethernetFrame) => frame.dst != MY_MAC_ADDRESS)
  .sendFrame()

```

shown in Listing 6.7). Then, the notification will be displayed to the user. Thus, the device can remain in standby mode, but the notification will nevertheless be distributed, which reduces power consumption on passive devices.

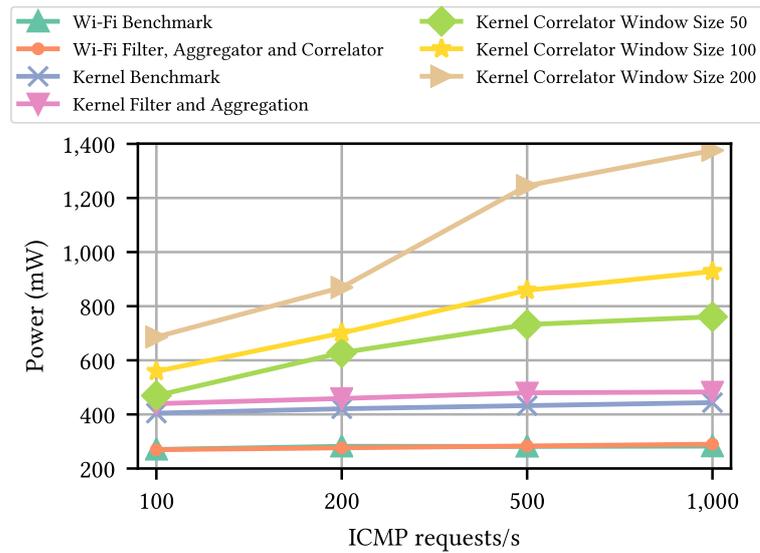
Another example of a mobile app is a local messaging app that enables attendees of such events to communicate locally via directly addressing devices. Other devices that are not the destination relay a message silently without involving the main CPU (lines 13 to 16), which again results in saving power on the passive nodes.

## Experimental Setup

To evaluate the concept of a virtual network stack on the Wi-Fi chip, we used ICMP echo packets for simulating such a network, since we could then compare the ReactiFi program with an in-kernel implementation. We measured the latency and power consumption for ICMP echo-request and echo-reply round trips implemented with ReactiFi on the Wi-Fi firmware. We used two Nexus 5 smartphones that were about 30 cm apart from each other. During the experiments, MAC Protocol Data Unit Aggregation (A-MPDU) and frame retransmission were disabled in the Wi-Fi firmware. Thus, every packet was sent once in a separate Wi-Fi frame. With this setup, we ensured that time measurements were reproducible and comparable.

Furthermore, we checked already used Wi-Fi channels from other networks to minimize interferences with other Wi-Fi networks. To be able to evaluate our framework on the basis of individual packets, we used ICMP echo-request and echo-reply packets with a total packet size of 1,200 bytes.

The ICMP echo-request and echo-reply round trip were implemented with three reactives. These reactives have in common that they all respond to an incoming ICMP echo-request packet with an ICMP echo-reply packet, but each reactive performs additional elementary computations. The first reactive is a simple packet filter that checks the signal strength for every incoming ICMP packet. The second reactive aggregates the signal strength for a certain number of packets. The third reactive determines the correlation between the size of the data of the `sk_buff` (i.e.,



**Figure 6.3.10** Power consumption: Wi-Fi and OS kernel.

the size of the packet payload) and the size of the entire Wi-Fi frame. For our aggregation and correlation tests, we used different buffer sizes. All tests were performed with 100, 200, 500 and 1,000 ICMP echo-requests per second.

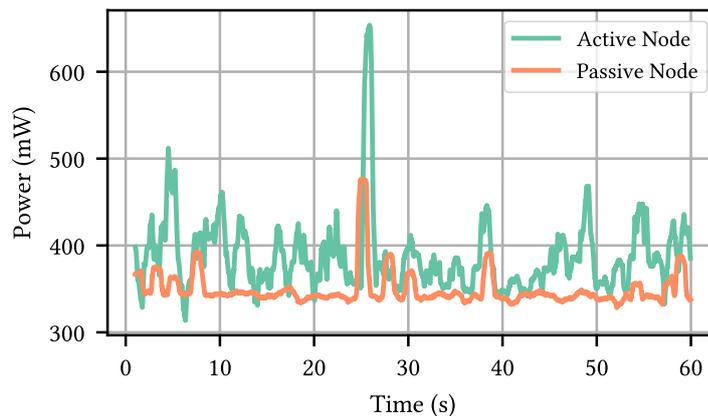
To show the benefits of executing code in the Wi-Fi firmware, additional tests were performed with the same reactivities executed in the operating system kernel.

### Power Consumption

The power consumption in our tests on the Wi-Fi chip was always between 265 mW and 309 mW, regardless of the kind of the test. Thus, for the Wi-Fi tests in Figure 6.3.10, only the Wi-Fi Benchmark and the mean power for all other tests, denoted as Wi-Fi Filter, Aggregator and Correlator, were plotted.

The power consumption in our OS kernel tests varies from 404 mW (baseline) and 1375 mW (correlation with 200 elements). In general, the kernel requires between 34% more power for the baseline and 60% more power for correlations with 50 elements than the Wi-Fi chip. In the kernel tests, correlations require more power than in other tests, due to the higher computational overhead. A filter has a complexity of  $O(1)$ , an aggregation  $O(n)$ , and a correlation  $O(n^2)$ , which was confirmed by the power consumption measurements. The higher complexity of correlations is the reason why the Wi-Fi chip could not successfully perform the two correlation tests with buffer sizes of 100 and 200 elements. Here, the limitations of the Wi-Fi chip become evident. The Nexus 5's ARM Cortex R4 Wi-Fi chip is a low-power co-processor for handling lightweight tasks. Thus, too complex or long-running tasks result in crashing the Wi-Fi firmware.

The power consumption of the on-chip tests shown in Figure 6.3.10 is comparable to the idle power in ad hoc mode and monitor mode (Figure 6.3.8). The kernel tests show a wider range of consumed power. The utilization is about as high as handing packets over to user space, but far away from the values the device consumes in total, as indicated in Figure 6.3.8.



**Figure 6.3.11** Power consumption of active and passive devices in the virtual network stack use case.

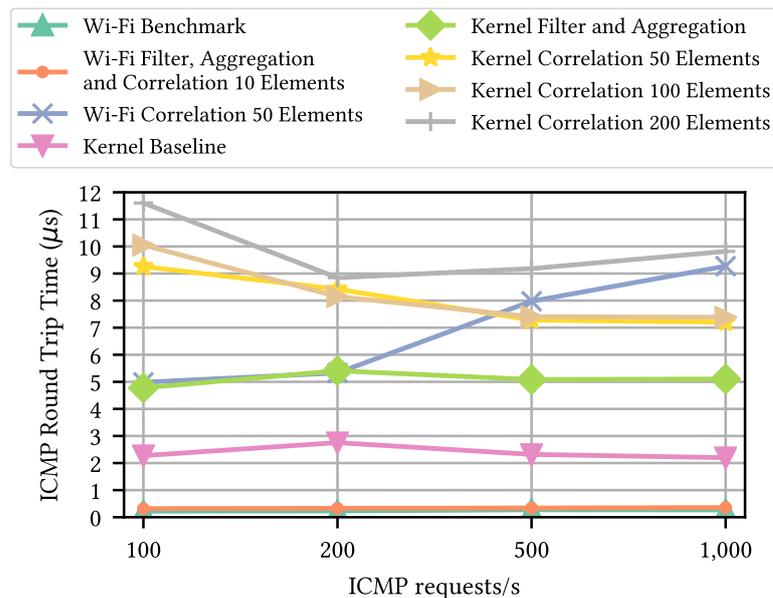
These experiments show that the potential power savings can be up to 60% if code is executed on the Wi-Fi chip instead of in the kernel, i.e., the battery of the Nexus 5 with a capacity of 2300 mAh would last about 12 to 19 hours longer, depending on the test. Additionally, the consumed power is always around 260 mW and 309 mW, whereas the kernel shows a much higher variance due to kernel specific behavior (e.g., context switches), which consumes additional power. On the other hand, the Wi-Fi chip is limited in its capabilities, as indicated by our correlation tests with larger buffer sizes.

In a second test setup, a third Nexus 5 smartphone was used as a passive relay node only forwarding packets without involving the main CPU. In this experiment, only a relatively small number of 100 requests/s were sent. As shown in Figure 6.3.11, the average power consumption of a passive device that only relays packets is about 350 mW. The active node that additionally builds packets on the Wi-Fi chip has an average power consumption of 390 mW. As already indicated in Figure 6.3.10, executing the same program in the OS kernel would lead to a power consumption of up to 700 mW for both active and passive devices. This indicates that executing an opportunistic networking protocol on the Wi-Fi chip reduces power consumption significantly.

Unfortunately, the Wi-Fi chip's timer has only a resolution of one millisecond, which is not sufficient to measure the execution time of simple reactivities like a packet filter. Furthermore, time measurements in the experimental code could disturb the experiments. To get high-resolution time measurements without interferences, a MacBook Pro was used to capture ICMP packets between the two Nexus 5 smartphones with Wireshark<sup>13</sup>. The round trip time (RTT) was calculated as the difference between the timestamps of the occurrence of the echo-request and the corresponding echo-reply seen in the Wireshark capture.

Figure 6.3.12 shows the round trip time of ICMP echo-requests and the corresponding echo-replies. Since the time was measured using an external sniffer, the RTT also includes the time the reactivities needed for the execution of the test. RTTs for tests executed on the Wi-Fi chip are always low at about 0.3 ms for the baseline, filter, aggregation, and correlation with a buffer size of 10 elements. Thus, filter, aggregation and correlation with 10 elements are plotted in the same graph in Figure 6.3.12. Correlations with 50 elements behave differently. While the Wi-Fi

<sup>13</sup><http://www.wireshark.org>



**Figure 6.3.12** ICMP round trip time: Wi-Fi and OS kernel.

chip can handle 100 and 200 requests/s in about 5 ms per packet, it takes about 8 ms for 500 requests/s and 9.3 ms for 1,000 requests/s.

### Round Trip Time

The kernel requires about 2.3 ms to 2.9 ms for handling ICMP echo-requests without any additional computations introduced by reactivities, which is an increase of 87% compared to the Wi-Fi chip. Filter, all aggregations and correlations with a buffer size of 10 elements require about 5 ms when executed in the kernel, which is 94% slower than on the Wi-Fi chip. Correlations with buffer sizes of 50, 100 and 200 elements require between 7 ms and 10 ms, depending on buffer size. While the Wi-Fi chip is faster in handling correlations with 50 elements at 100 and 200 requests/s (about 5 ms compared to 8 ms, which is about 38% faster), the kernel is about 9% faster in executing the correlation tests at 500 requests/s and 22% faster at 1,000 requests/s. Due to the fact that for low packet rates the kernel falls into power saving mode and needs to be woken up quite often, correlation experiments at low packet rates executed in kernel require more time than at higher packet rates. The number of sleep-wake cycles decreases with higher packet rates, thus the required time decreases. To summarize, executing code in the Wi-Fi chip can decrease the runtimes up to 94%, but the Wi-Fi chip cannot execute arbitrary code and is limited in its computational capability and memory capacity. Thus, Wi-Fi chip execution is particularly useful for resource-constrained, low-latency applications.

Finally, we could not see any packet loss, neither in tests on the Wi-Fi chip, nor in kernel. The only exception were correlations of 100 elements, where the Wi-Fi chip could not handle the amount of data and dropped the packets.

### 6.3.5 Related Work

In the following, related work is discussed. ReactiFi is related to programmable Wi-Fi firmware (Section 6.3.5.1), software-defined wireless networking (Section 6.3.5.2), and embedded reactive programming (Section 6.3.5.3).

#### 6.3.5.1 Programmable Wi-Fi Firmware

Tinnirello et al. [Tin+12] present a finite state machine approach for defining and implementing MAC protocols on Wi-Fi firmware, and Bianchi et al. [Bia+12] extend this approach by introducing MAClets for simplifying the programmability of MAC protocols executed on Wi-Fi firmware. In contrast, ReactiFi is not limited to MAC protocols, but supports a wide range of applications running on a Wi-Fi chip. Furthermore, ReactiFi runs on off-the-shelf mobile devices and optimizes various metrics on end devices, whereas MAClets require a software-defined radio platform (USRP B200) for execution.

#### 6.3.5.2 Software-defined Wireless Networking

Software-defined networking (SDN) [Kre+15; Nun+14; KF13] supports programmable network behavior in a centrally controlled manner to facilitate flexible network management. SDN separates forwarding of data packets in the data plane from routing in the control plane. In addition to routing, SDN in wired networks typically addresses data processing functionality in higher networking layers. Network programmability and centralized control are attractive means for managing wireless networks, where the paradigm is referred to as software-defined wireless networking (SDWN) [Cos+12; Ber+14].

While aspects such as routing, security, and network slicing may profit from network programmability in general, 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 [Fon+17; Ku+14]. Schulz-Zander et al. have proposed OpenSDWN, an approach based on SDWN and Network Function Virtualization (NFV) [Sch+15a; Sch+15b; Sch+17]. 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. [Hät+16] 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 firmware on off-the-shelf mobile devices, as it is supported by ReactiFi.

#### 6.3.5.3 Embedded Reactive Programming

Reactive programming for resource-constrained Wi-Fi firmware is related to other programming models for embedded systems. TinyOS [Lev+05] is a scheduler and a collection of drivers for low-power wireless embedded systems. It allows event-driven programming with nesC [Gay+14], a C language-derivate, which has a few similarities with ReactiFi: nesC programs are assembled from components that run concurrently in the form of run-to-completion tasks. In ReactiFi, reactive

programs are structured into reactives that are scheduled non-preemptively. In contrast to nesC, reactives are not statically linked to each other. This supports dynamic modification of the data flow, which combines features from imperative and functional programming. RIOT OS [Bac+13] 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. Context switches can occur either on hardware interrupts, voluntarily or during a blocking operation. Data flows between program components are not integrated. They would have to be realized explicitly by means of inter-process communication. In contrast, automatic resolution of dataflow dependencies between reactives is a feature of ReactiFi by design. Furthermore, RIOT OS and Tiny OS have not been implemented as runtime environments on Wi-Fi firmware.

### 6.3.6 Summary

In this section, ReactiFi was presented, which facilitates reactive programming of Wi-Fi firmware on mobile devices. In ReactiFi, application programmers develop reactive programs in a language based on TypeScript to operate on events at different layers of the networking stack. Thus, reactives can be used to monitor radio links, process frames/packets, and trigger actions in Wi-Fi firmware. Reactive programs written in ReactiFi are transpiled and compiled to Wi-Fi firmware patches that are dynamically linked during runtime. Thus, PHY layer events involving SNR and MAC layer events related to distribution system values can be processed on the Wi-Fi chip without involving the main CPU.

ReactiFi realizes transitions between modes of reactives (CT 3). The execution of reactive programs on the Wi-Fi chip, instead of running them on the CPU, is particularly useful for resource-constrained, low-latency applications.

The benefits of ReactiFi were demonstrated in two use cases. In a nearby video streaming use case, in which the Wi-Fi mode was dynamically switched between 802.11n and 802.11z TDLS, throughput and video quality could be increased significantly. In a second use case, a virtual network stack using ReactiFi was implemented on the Wi-Fi chip, allowing programmers to develop opportunistic networking applications while a mobile device is still connected to Wi-Fi APs, and improves power consumption and latency considerably.

## 6.4 Summary

This chapter provided two instances of transitions between modes in near-\* computing. This type of computational transition leverages the full potential of mobile hardware/software architectures by performing parts of network- and application services on-device 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).

In particular, the following contributions were described:

- Multimodal CEP realizes transitions between modes of CEP operators (CT 3). It automatically breaks up CEP queries and selects the most adequate execution mode for the involved CEP operators. Filter, aggregation, and correlation operators can be expressed in a high-level language without requiring system-level domain-specific knowledge.
- Reactive Programming of Wi-Fi Firmware on Mobile Devices realizes transitions between modes of reactivities (CT 3). The execution of reactive programs on the Wi-Fi chip, instead of running them on the CPU, is particularly useful for resource-constrained, low-latency applications.

These realizations reduced power consumption and improved latency of several network and application services considerably.

There are several areas for future work. For example, multimodality could be introduced to support computing-intensive functions like in-network machine learning or image processing, bypassing the main CPU. It would also be interesting to evaluate ReactiFi in other usage scenarios based on 60 Ghz Wi-Fi networks, or transferring it to other wireless technologies, such as WiMAX, 4G/5G, LoRa, Bluetooth, or Zigbee.



# 7

## Transitional Near-\* Computing for Emergency Response Applications

In this chapter, the concept of transitional near-\* computing is applied to support rescuers and affected persons during an emergency event with adaptive emergency response applications. Section 7.1 describes basic requirements of emergency response applications and the benefits of transitional near-\* computing in such a scenario. Section 7.2 discusses the design of an adaptive emergency response application that helps affected people and rescue teams to find missing people while conserving mission-critical battery power and network bandwidth. Section 7.3 gives an overview of the computational transitions applied in this application. Sections 7.4–7.6 describe implementation and experimental results the emergency response application. Finally, Section 7.7 summarizes the chapter.

*Parts of this chapter have been published in [Gra+17; Gra+18b; Gra+18a; Ste+].*

### 7.1 Motivation

Emergency and disaster management is a crucial task for international organizations like the World Health Organization (WHO) and for governments around the world. During and shortly after an emergency event, such as an earthquake or a hurricane, it is essential to maintain critical infrastructures like power stations, water reservoirs, and telecommunication centers while taking immediate measures to prevent the worsening of the situation to a disaster.

“

**Disaster** describes a serious disruption of the functioning of a community or a society causing widespread human, material, economic, or environmental losses which exceed the ability of the affected community or society to cope using its own resources.

*Definitions of the World Health Organization<sup>1</sup>* ”

In order to respond quickly after an emergency event and to prevent it from turning into a disaster, international organizations and governments need to take managerial measures for different disciplines or phases of emergency management: Mitigation, Preparedness, Response, and Recovery [HBC17]. The disciplines of mitigation and recovery focus on restoring from and preparing for future hazards like floods or earthquakes. Response describes the first-aiders’

<sup>1</sup><http://www.who.int/hac/about/definitions/en>

responsibilities like rescuing injured persons or suppressing fires. Preparedness is defined as the state of readiness to respond to any kind of emergency.

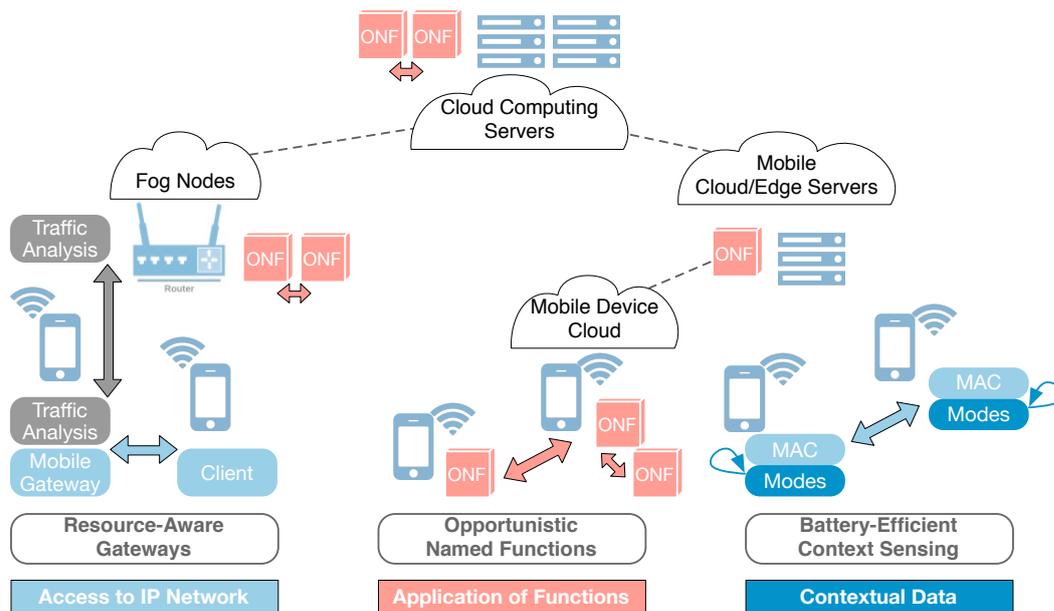
For both preparedness and response to an emergency event, information about the current situation is crucial for affected people and rescue teams, but telephone lines, cellular base stations, and parts of the Internet backbone might be destroyed, disrupted, or overloaded due to network congestion. Therefore, network operators need to solve challenges such as high network latencies, limited network bandwidth, limited coverage areas of wireless networks, as well as the lifetime of generator-powered gateways and battery-powered mobile devices and sensors. To meet these challenges and to provide situational awareness, emergency and disaster response applications have been built on top of network infrastructures that are resilient to a breakdown or segmentation of a centralized IP network, such as mobile ad-hoc networks (e.g., [OLG10]) and delay-tolerant networks (e.g., [Bau+16]). Computational resources in this scenario have been provided with computing capabilities based on mobile cloud computing [Shi+18], mobile edge computing [Sat17], and fog computing [Sap+16]. In fact, near-\* computing environments are increasingly used to provide computing capabilities for latency-critical responses and to extract information near-user, near-network, and near-data.

Transitional near-\* computing, however, can strengthen the adaptivity of emergency response applications, with respect to the quickly changing network infrastructures and computing resources during an emergency event. Furthermore, it can address novel challenges resulting from the enormous growth of data volume and number of data sources during an emergency, such as the growing number of mobile and IoT devices that need to process information quickly for situational analysis, without draining resources from other parts of the critical infrastructure. In particular, energy-efficient transitional near-\* computing can save power during an emergency event, which is an operation-critical resource for battery-constrained sensors, mobile devices, and generator-powered stationary servers.

### 7.2 Design

In challenging environments with network overload, battery-constrained mobile devices, and disrupted Internet services, transitional computing can be applied to provide basic means of communication and computation. The following challenges need to be addressed by an adaptive emergency response application.

- 1. Overload of Undamaged Parts of the Network Infrastructure:** Affected people and rescue teams make heavy use of the parts of the IP network that are undamaged. They use normal means of communication, such as messaging services, social networks, and others. Non-priority traffic might overload the limited network bandwidth, disrupting essential means of communications.
- 2. Battery-constrained Devices:** Without a working power grid that can be used to charge battery-powered devices, affected people depend on their remaining battery levels. Therefore, increasing the battery lifetime by saving battery power is a crucial task in an emergency scenario.
- 3. Disrupted Internet Services:** Internet and cellular network based services such



**Figure 7.1** Transitional near-\* computing for emergency response applications.

as localization services or cloud computing services might not be accessible in an emergency scenario. These kinds of services need to be re-established in an ad-hoc and distributed manner, leveraging the storage and computation capabilities of the remaining battery-powered edge devices.

These challenges are addressed by the following solutions based on transitional near-\* computing.

- S 1 Dynamically Placed Resource-aware Gateways:** Unaffected multi-homed mobile phones with a working Wi-Fi or cellular network connection can be dynamically transformed into mobile gateways, granting Internet access to nearby wireless devices through Wi-Fi tethering or -bridging. Mobile devices are put into a gateway role depending on their remaining resources, i.e., their available bandwidth to the IP network infrastructure or their battery levels. Mobile applications can be blocked at the IP layer and hosts at the MAC layer in order to save network bandwidth and to avoid network overload. MAC- and IP-layer blacklists are generated on demand by a network monitoring tool that is dynamically placed within the network.
- S 2 Opportunistic Named Functions in a Disruption-tolerant Network:** Named functions are applied on content, which is distributed via network such as continuous sensor data or photos taken with a smartphone camera. Different implementation variants can be declared as alternatives, such that a device can select the best alternative according to its context. In particular, this solution is designed to support affected people and rescue teams in the search for missing persons.
- S 3 Battery-efficient Context Sensing in Mobile Devices:** Information about the context [Bel+12] of the devices and their users, including user activities and location, is

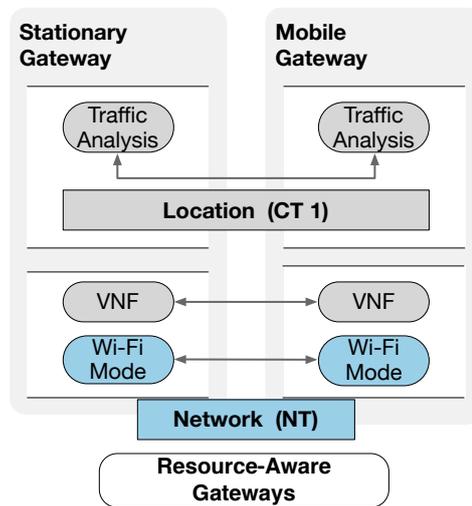
important for rescue operations. Instead of continuously monitoring every sensor of a mobile device, mobile phones are put to deep sleep (dozing) until a relevant event occurs. Apart from that, affected people within buildings might not be able to receive a GPS signal to determine their position. Therefore, a low-power in-door localization service is designed using the relative received signal strength (RSSI) of 802.11 Wi-Fi frames on the Wi-Fi chip of mobile devices.

In Figure 7.1, the relation between the solutions is shown. Dynamically placed resource-aware gateways (Solution 1) permit access to working cellular- and Wi-Fi networks that are primarily used for essential means of communication. Opportunistic named functions (Solution 2) enable applications to be executed on-device, at the network edge, or in the network core depending on the network connection and context of the involved devices. Battery-efficient context sensing (Solution 3) is used to provide context information about the context [Bel+12] of the devices and their users.

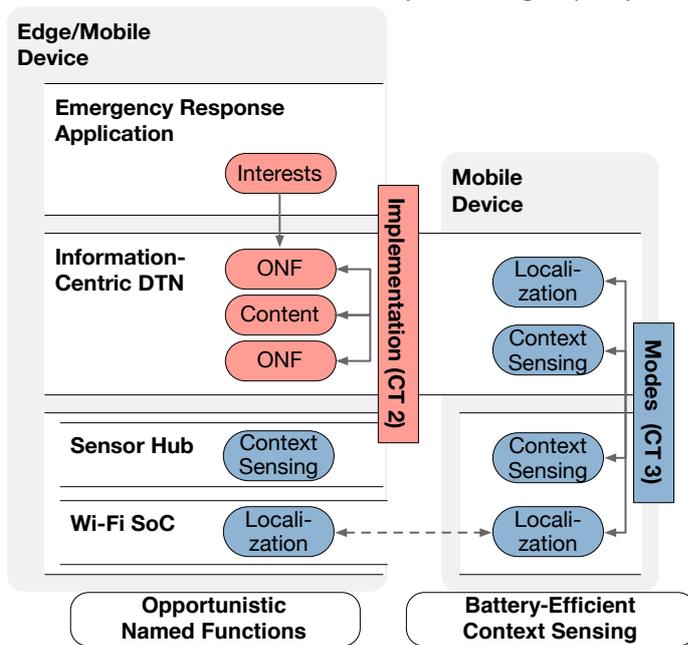
### 7.3 Computational Transitions

Figure 7.2 depicts the computational transitions between locations, implementations, modes, and network mechanisms within an adaptive emergency response application. At the bottom, dynamically placed resource-aware gateways (Solution 1) apply two kinds of computational transitions. In order to transform unaffected multi-homed mobile phones with a working Wi-Fi or cellular network connection into mobile gateways, network transitions (NT) are performed in combination with transitions between VNF locations (CT 1).

Furthermore, at the top left of Figure 7.2b, transitions between alternative opportunistic named functions (CT 2) are performed to execute applications on-device (Solution 2). The selection between alternative functions is made according to the device's context. The context is sensed battery-efficiently (Solution 3) on-device. To achieve this, transitions between modes of context sensors and in-door localization protocols (CT 2) are performed.



(a) Transitions between locations within an adaptive emergency response application.



(b) Transitions between implementations, modes, and network mechanisms within an adaptive emergency response application.

**Figure 7.2** Transitions within an Adaptive Emergency Response Application.

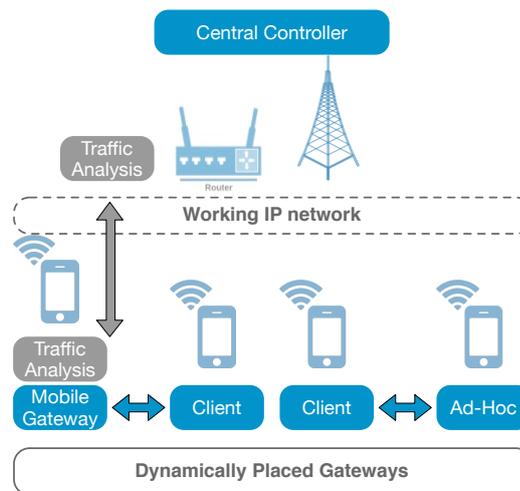


Figure 7.3 Design of dynamically placed bandwidth-aware gateways.

## 7.4 Dynamically Placed Resource-aware Gateways

Social media and communication via social network applications are becoming even more important during an emergency event. For example, during the big flood in the U.S. state of Louisiana in 2016, the worst disaster in the U.S. since Hurricane Sandy in 2012, flood inundation maps, locations of emergency shelters, and other information were actively shared online via Facebook [KH18]. This is not a singular case. In general, typical mobile social network applications play an important role in disaster management, particularly in sharing emergency information [Mic14]. These applications rely on a working IP-based network infrastructure. Resource-aware gateways allow a large number of affected people to access this infrastructure and to use these services, without overloading the network and draining their devices' batteries.

Figure 7.3 depicts the design of our application. It is based on dynamic role assignments (Section 4.3) in an IP-based wireless network consisting of multi-homed mobile devices and access points connected to those parts of the IP network infrastructure that are still working. According to the concept of dynamic role assignments, the assignment of a role is controlled by a central controller. Four different roles are used in this design: (i) Gateway role, essentially turning a mobile device's Wi-Fi network interface card into access point mode that routes incoming traffic to another network, (ii) ad-hoc role, spanning a wireless ad-hoc network without a centralized topology, (iii) traffic analyzer role, essentially a virtual network function that applies rules to the flow of network packets and extracts MAC and IP layer filters, (iv) client role, which accesses the working IP infrastructure via gateways. Transitions between these roles, i.e., transitions between the location of network services, are the technical basis for balancing load and battery consumption between mobile devices, while improving the accessibility of the working IP network infrastructure in an emergency event.

While the evaluation of both access point and ad-hoc role have already been described in Section 4.3, the traffic analyzer role is specific to this application and is detailed in the following. To realize the traffic analyzer role, the intrusion prevention system (IPS) `snort`<sup>2</sup> is used. In our experimental setup, the IPS system is configured with rules to detect flows of network traffic

<sup>2</sup><https://www.snort.org>

**Table 7.1** IPS setup measurement results.

Experiment	Mean Latency	Mean Throughput	Mean Power
idle, w/o IPS	/	/	1.572 W
idle, w/ IPS	/	/	1.576 W
iperf, w/o IPS	8.29 ms	44.65 Mbit/s	2.095 W
iperf, w/ IPS	70.2 ms	44.10 Mbit/s	2.225 W

and start dropping packets if the packets per second metric reach a certain threshold. This filter is realized with `iptables` rules at the IP layer. When the system detects a certain number of flows exceeding the bandwidth threshold, the IP layer filters are removed and, instead, the source host is blacklisted at MAC layer using the blacklist capability of `hostapd`.

In Table 7.1, the experimental results for the traffic analyzer role are shown. The experiments were conducted on a dual-homed Raspberry Pi 3, routing between two Wi-Fi subnets. First, the idle power consumption is measured with and without an activated IPS. Without any traffic, the IPS does not introduce a power overhead. In contrast, during throughput measurements with `iperf3`, the IPS increases the power consumption by 5.5%. In this experiment, the CPU dominates the power consumption, especially during the positive detection of blacklisted traffic. The additional involvement of the CPU that is needed to inspect incoming packets introduces significant power overhead. Moreover, this packet inspection introduces a noticeable increase of the network latency (8.5x). Filters on MAC and IP layer do not introduce a significant power overhead.

## 7.5 Search for Missing Persons

During an emergency event, the search for missing persons is a primary task for rescue teams and highly important for affected people. Facebook<sup>3</sup> and Google<sup>4</sup> provide interfaces that help affected people to reconnect with friends and relatives in the aftermath of a disaster event. While the network infrastructure for these IP-based services was addressed in the previous Section 7.4, this section describes an application that supports the search for missing persons in an emergency event even though parts of the infrastructure, i.e., telephone lines, cellular base stations, are damaged, disrupted, or overloaded due to network congestion. For this application, the remaining communication infrastructure, such as battery-powered sensors, mobile devices, and generator-powered routers are used to establish a resilient disaster-response communication system.

The application for the search for missing people in a disruption-tolerant network is realized with Opportunistic Named Functions (ONFs) in a DTN network (see Section 5.3). In general, ONFs allow to reflect inherent tradeoffs between either spending CPU cycles and battery power for function execution, or spending battery power and air time for data transmissions. In case of our application, any rescuer of an affected person might be interested in all detected faces in all pictures taken by the mobile devices' cameras that are published within the network. But simply transferring all high-resolution raw images taken with mobile phone cameras from the content producer to the consumer risks congesting the network. ONFs, in contrast, can be used

<sup>3</sup><https://www.facebook.com/about/safetycheck>

<sup>4</sup><https://google.org/personfinder/global/home.html>

**Table 7.2** Search for Missing Persons: ONFs  
Opportunistic Named Functions (ONFs) for adaptive in-network processing in a disruption-tolerant network.

ONF	Input	Output
/camera	/	Photos taken with the camera of content producer.
/face_detection	Image	Parts of the image containing detected faces and bounding boxes according to [Lam+17].
/gray	Image	A version of the input transformed to grayscale.
/scale	Image	A down-scaled version of the input.
/black_white	Image	A black/white version of the input.

to perform transitions between implementations of functions that reduce either the amount of data to be transferred or save battery of the devices. More specifically, ONFs are used to (i) delegate function execution, (ii) apply multiple stages of image preprocessing that can be deployed even on small devices and can be executed in a pipeline, and (iii) prioritize processed content based on the importance for a content consumer.

In the following, the realization of the application with the help of ONFs is described. Table 7.2 defines the interests specified by our application. The application relies on face detection algorithms that are applied to pictures taken by visual sensors and mobile users. The algorithms are applied either in their original color or transformed to a grayscale image, and low-resolution black/white images, i.e., the application visualizes the information to the user by showing detected faces in high resolution, with a low-resolution black/white-underlay for situation analysis. For this application, both variants need to be processed and transmitted, since it is not possible to perform face detection algorithms on black/white images with accurate results [Lam+17].

A different approach would be to detect the faces on the producers' nodes and to send only those image areas containing detected faces that are significantly smaller than the original image. However, since face detection is not delegated, this approach holds the risk that in some cases (i) the workload within the network is not shared across all nodes, because very active content producers drain their batteries faster than others, and (ii) images are transferred in raw format, since producers like microcontroller-based cameras cannot execute these algorithms. Moreover, there are several implementations and concepts in the domain of face detection algorithms, and a user can choose between different concepts, their implementations, and configurations, which are different with respect to execution time, resource consumption, and quality of the results. By concatenating different implementations of face detection algorithms in a pipeline where each stage can be executed on a different node, we are able to achieve preliminary results with less resource-hungry face-detection algorithms, thereby reducing the transmission time for the following hops.

In Table 7.2, the ONFs for an application to search for missing persons are specified. Depending on the type of the processor, basic transformations on /camera pictures can be applied. Internet-of-Things (IoT) based wireless vision sensors usually run on digital signal processors (DSPs) that can at least perform compression and color filters. In our scenario, the ONFs can transform pictures to grayscale (/gray), shrink the picture (/scale) and generate a black/white version of the image (/black\_white). Since these are inexpensive transformations, they are applied to reduce the amount of transferred data in order to avoid network congestion.

**Table 7.3** Search for Missing Persons: Interests  
Specified interests in named content.

<b>Interest</b>	<b>Description</b>
<code>/camera/face_detection</code>	Alternative A: perform a face detection on the original resolution (8-16 megapixels).
<code>/camera/gray/face_detection</code>	Alternative B: perform a face detection on a picture with reduced color depth.
<code>/camera/scale/black_white</code>	A low-resolution black/white-underlay as an addition to the face detection.

Table 7.3 shows the interests in named content specified by our application. A face detection can be performed either directly on an image in the original resolution, or on pictures with reduced color depth. As described above, a black-/white-underlay for the detected faces is also provided. The processing of these interests in named content within a disruption-tolerant emergency network is illustrated in Figure 7.4. It is assumed that all nodes already received the interest packets sent by the consumer. The producer at the bottom of the figure is a microcontroller-based camera, which is able to make basic image transformations, such as transforming to grayscale and shrinking a black/white version of an image. The first node at the top of the picture is a command center or the head of a rescue team that relies on information about missing persons. In this illustration, a 11 megapixel picture (2.9 MB) is taken by the content producer's camera. It is then reduced to a 2.4 MB grayscale and a 130.5 KB downscaled b/w image. In a wireless network with end-to-end connectivity, it is feasible to transfer only one version of the image from the producer to the consumer. However, in a DTN with no or with an unstable end-to-end connectivity, the probability of successfully passing information between nodes increases with smaller content. This is especially true for Low-Power Wide Area Networks (LPWAN), where LPWAN devices trade their power efficiency and high range (between 2 km and 10 km) with extremely low bandwidth (1-25 kbps). In this illustration, the resulting images are transferred to two different nodes: an ARM-based smartphone that can perform a face detection algorithm and a microcontroller-based Low Power Wide Area Network (LPWAN) device. Due to its low bandwidth, the microcontroller-based LPWAN device cannot transfer a 2.4 MB image with 25 kbps in reasonable time. On the other hand, the mobile phone can perform a face detection on the grayscale image. It can extract 3 faces (0.1 MB, 0.23 MB and 0.12 MB), which are then transmitted via 802.11 Wi-Fi. At the content consumer, the scaled black/white image and each of the three detected faces arrive separately. At each arrival, the user is notified and supplied with an image that consists of a low-resolution black/white image in the background and grayscale faces in the foreground, which can be helpful for situation-awareness.

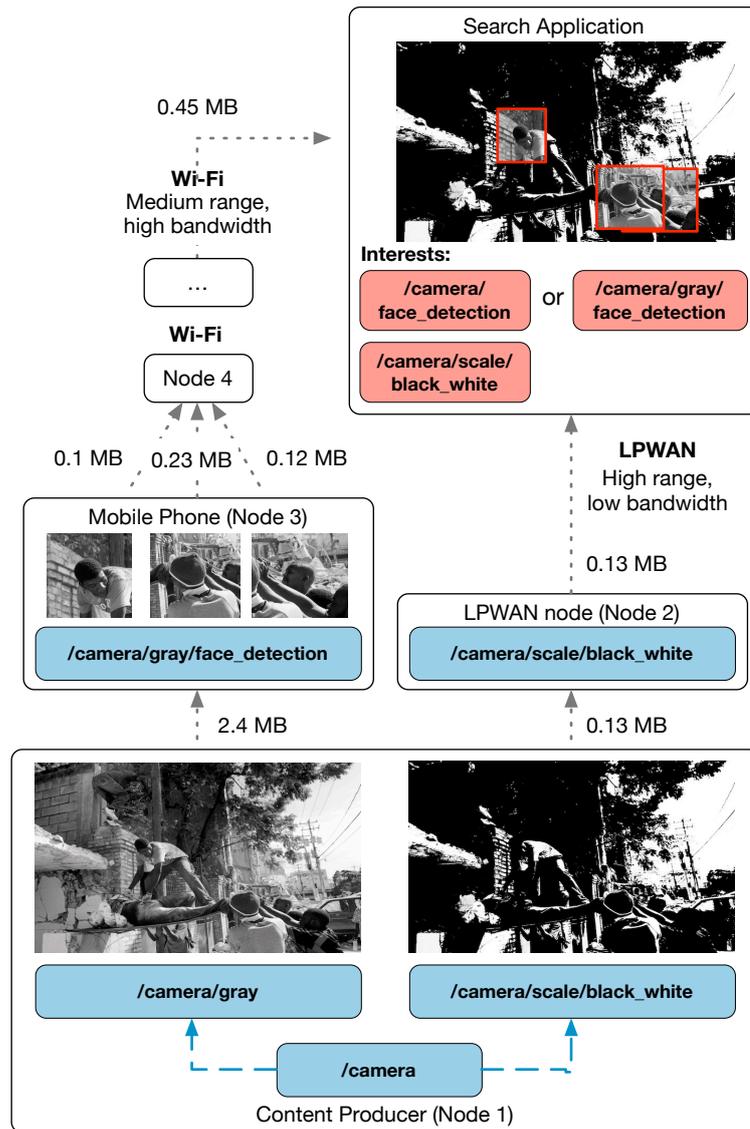


Figure 7.4 Search for missing people in a disaster scenario.

## 7.6 Battery-efficient Context Sensing in Mobile Devices

During an emergency event, information about the context [Bel+12] of the devices and their users is also used to support rescuers and affected people. Localizing victims in real time, e.g., by deploying sensors [YSG17] or leveraging the Wi-Fi Received Signal Strength (RSS) [LJJ17], can be crucial for their rescue, and gathering information about life signals and the victim's activities might provide essential information for first aid personnel. If possible, this information can be either transferred to a network and used for situational analyses, or stored on local storage for later investigations. In the following, context information is gathered battery-efficiently during an emergency event. To achieve this, transitions between modes of context sensors and in-door localization protocols are performed. In Section 7.6.1, a battery-efficient activity detection approach is described. In Section 7.6.2, a low-power in-door localization service is presented.

### 7.6.1 Physical Activity Detection

To detect victim's activities to support first aid personnel, multimodal CEP on smartphones (see 6.2) is used. Multimodal CEP is able to facilitate continuous sensing on mobile devices in a battery-friendly manner by processing streams of events on-device 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).

- **Outdoor detection.** Outdoor detection can be used as a low-power alternative to a power-hungry GPS-based sensor or as an indicator to support indoor localization.
- **Physical inactivity.** Physical inactivity can be interpreted as a missing sign of life. This can be detected by aggregating the average movement of the accelerometer over a 10-minute-time window filtering the result as to whether the average value is less than a small threshold.
- **Phone close to person.** If a person loses his or her smartphone, he or she cannot be monitored. This can be detected by a filter evaluating the proximity sensor to determine whether the device is on the body or not.
- **Possible falls.** Falls are a major cause of fatal injuries and are the primary reason of injury-related deaths for seniors.

Listings 7.1, 7.2, 7.3 show the queries used for outdoor detection, phone-close-to-person detection, and fall detection. The outdoor detection in Listing 7.1 filters the ambient light and the magnetometer to check whether they exceed or fall below a threshold value [Zho+12]. The proximity detector checks the distance between the smartphone and the victim. Possible falls (Listing 7.3) are detected by combining pre-filtered events from the accelerometer and the barometer, within a small window (the 2 most recent events). The applied filters look for fast accelerations, similar to a free fall, and decreased air-pressure, representing a downward movement, respectively [Che+12]. Instead of continuously monitoring every sensor of a mobile device, they are put to deep sleep (dozing) until a relevant event occurs, which is detected with on-chip capabilities without involving the mobile CPU.

**Listing 7.1** Outdoor detection query.

```
SELECT * FROM PROXIMITY WINDOW(COUNT 2) P,
        MAGNET WINDOW(COUNT 2) M
WHERE SQRT(M.X^2 + M.Y^2 + M.Z^2) < m_threshold AND
      P.AMBIENT > a_threshold;
```

**Listing 7.2** Phone close to person.

```
SELECT * FROM PROXIMITY P
WHERE P.PROXIMITY > a_threshold;
```

**Listing 7.3** Fall detection query.

```
SELECT * FROM BAROMETER WINDOW(COUNT 2 JUMP 2) B,
        ACCELEROMETER WINDOW(COUNT 2 JUMP 2) A
WHERE B.PRESSURE < b_threshold AND
      SQRT(A.X^2 + A.Y^2 + A.Z^2) > a_threshold;
```

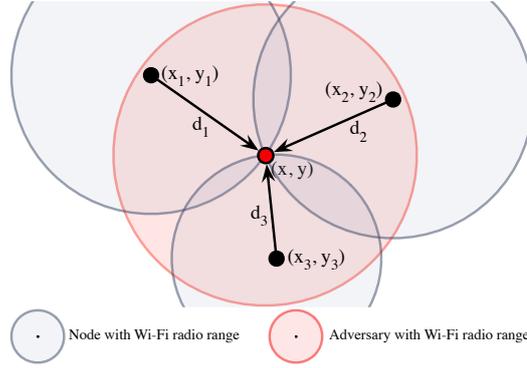
**Table 7.4** RAM usage of the activity detection queries.

Query	Total Used RAM (Bytes)
Outdoor	317
Physical Inactivity	288
Phone Proximity	187
Fall detection	354
All	1146

The activity detectors have been evaluated on a Nexus 5 smartphone using two different sensor hub realizations (ATmega328 and ATmega2560) for multimodal CEP and then compared to executing it in user mode. As discussed in Section 6.2.4, the ATmega328 executes queries with a better battery efficiency than the ATmega2560, but has a smaller memory capacity. Table 7.4 shows the used RAM in bytes for each query. Note that combining all queries does not result in the arithmetical sum because multiple queries share the sensor library code. The largest sensor library is the GPS module library with a size of 544 bytes. The execution of the hub mode operators takes about 160 milliseconds, resulting in 6 iterations per second. The ATmega2560 consumes 617.5 mW of power during the application. Reduced by the static power consumption stemming from the sensors and the measurement environment, the power consumption is about 294.5 mW, which is 61.8% of the power consumption of performing the queries in user mode, i.e., power savings of 38.2% are obtained.

### 7.6.2 Indoor Localization of Mobile Devices

To localize affected people within buildings that might not be able to receive a GPS signal to determine their position, a trilateration approach using the relative received signal strength (RSSI) of 802.11 Wi-Fi frames with the help of reactive Wi-Fi programming (see Section 6.3) is realized. Trilateration is a technique commonly used to locate and track people in wireless scenarios [DA18; BP00]. Together with path loss and shadowing models, the RSSI can provide a



**Figure 7.5** Visualization of the trilateration approach with at least three known distances  $d_i$  and points  $(x_i, y_i)$ .

distance estimation. A commonly used model for estimating the distance based on the RSSI is

$$P(dBm) = A - 10n \log_{10}(d)$$

where  $P(dBm)$  denotes the path loss for a received signal strength of  $dBm$  at position  $d$ ,  $A$  the received power in  $dBm$  at a known distance, and  $n$  the path loss exponent. Given this equation, the distance  $d$  can be calculated as

$$d = 10^{\frac{A - P(dBm)}{10n}}$$

for known  $A$  and  $n$ . Retrieving  $A$  can be achieved by measuring the RSSI over a sufficient time period like 10 minutes at a distance of 1 m and averaging the RSSI values. Usually the parameter  $n$  is within the interval  $[2, 4]$ , where the lower bound 2 is used outdoors in free space and 4 in lossy environments within buildings.

Given at least three distances  $d_i$  calculated with the equation above and three corresponding known points  $(x_i, y_i)$  to a victim, the victim's position  $(x, y)$  can be calculated as shown in Figure 7.5.

The three distances can be expressed as:

$$d_1 = \sqrt{(x_1 - x)^2 + (y_1 - y)^2} \quad (7.1)$$

$$d_2 = \sqrt{(x_2 - x)^2 + (y_2 - y)^2} \quad (7.2)$$

$$d_3 = \sqrt{(x_3 - x)^2 + (y_3 - y)^2} \quad (7.3)$$

With Equations (7.1), (7.2) and (7.3), a least squares approximation can be used, which gives us

$$\begin{bmatrix} 2(x_2 - x_1) & 2(y_2 - y_1) \\ 2(x_3 - x_1) & 2(y_3 - y_1) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} d_1^2 - r_2^2 + x_2^2 + y_2^2 - x_1^2 - y_1^2 \\ d_1^2 - r_3^2 + x_3^2 + y_3^2 - x_1^2 - y_1^2 \end{bmatrix} \quad (7.4)$$

With  $M$  and  $N$  defined as

$$M = \begin{bmatrix} 2(x_2 - x_1) & 2(y_2 - y_1) \\ 2(x_3 - x_1) & 2(y_3 - y_1) \end{bmatrix} \quad (7.5)$$

$$N = \begin{bmatrix} d_1^2 - r_2^2 + x_2^2 + y_2^2 - x_1^2 - y_1^2 \\ d_1^2 - r_3^2 + x_3^2 + y_3^2 - x_1^2 - y_1^2 \end{bmatrix} \quad (7.6)$$

the location of the target can be calculated with Equation (7.7)

$$\begin{bmatrix} x \\ y \end{bmatrix} = M^{-1}N \quad (7.7)$$

The trilateration approach was implemented in a distributed reactive graph containing multiple reactives to locate a victim, as shown in Listing 7.4. As soon as an identification request is sent by a participating node, each node in the network responds with the distances to one-hop neighbor nodes that have been previously calculated (lines 7 to 9). All the computed distances are broadcasted together with the own GPS position (which is transferred to the Wi-Fi chip via an ioctl) and MAC address. As soon as enough distances have been collected, the requesting node can trilaterate the position of every other node, especially the victim's position, which is shown in lines 11 to 16. This process can be repeated periodically, to track a victim if needed (not shown in the code).

**Listing 7.4** Reactive program for localizing victims.

```

var distances = RawStream
  .join(StatStream)
  .groupBy((frame) => frame.src)
  .timeWindow(5)
5   .map((frame) => get_distance(frame.rssi));

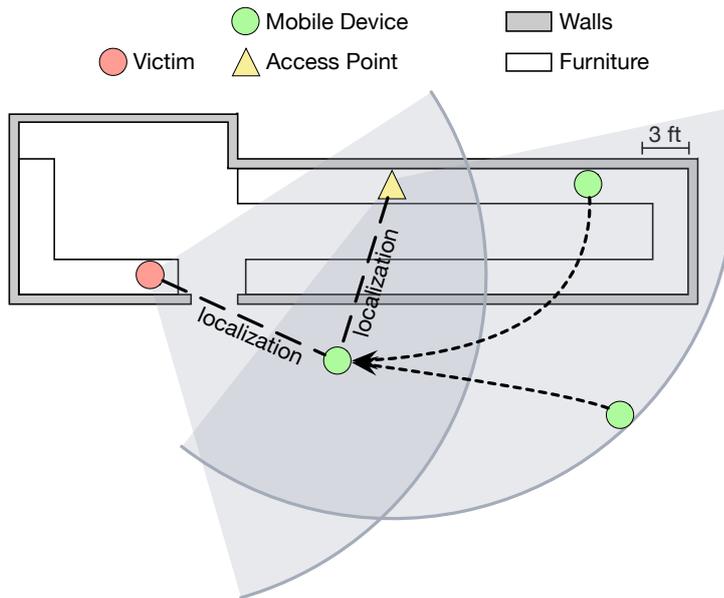
var requests = ReactiveStream
  .filter((packet) => packet.type == REQUEST)
  .send_distances();
10

var responses = ReactiveStream
  .filter(
    (packet) => packet.type == RESPONSE &&
      DISTANCES.length >= 3
15  )
  .trilaterate_and_identify();

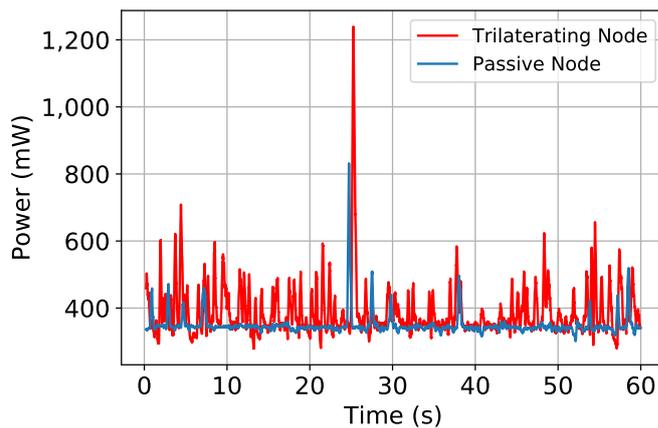
```

Although it is possible to obtain the RSSI value for each Wi-Fi frame in user space (or kernel), this is only achievable by actively pushing every single Wi-Fi frame to user space which increases power and computation overhead. In contrast, our distributed reactive Wi-Fi approach achieves trilaterating of victims passively without invoking the main CPUs of the mobile phones or relying on dedicated apps. All computations are entirely performed on the Wi-Fi chips, which results in lower power consumptions than executed on the main CPUs.

To evaluate this approach, three Nexus 5 mobile phones were used. Forming anchor points, the mobile phones sent the distances to all other devices within their respective ranges as soon as the trilateration request had been received. One of these Nexus 5 phones was used to initiate the trilateration process. For this purpose, a custom Wi-Fi frame was assembled containing the trilateration request. As soon as the other mobile phones had received the request, the distances were broadcast with a custom Wi-Fi frame through the network. The initiating node collected all distances and started the trilateration using the above method. As a target for the localization approach a Raspberry Pi was used. All devices were set in the wireless range of each other. Furthermore, the mobile phones were connected to an access point and the devices were put at fixed positions during this test. To be able to analyze the RSSI of all neighbors, the Nexus 5 phones were set to monitor mode. Since the described trilateration method requires



**Figure 7.6** Experimental setup of the in-door localization approach.



**Figure 7.7** Power consumption of active and passive devices during trilateration.

two parameters,  $A$  and  $n$ , these values had to be set. Our experiments were conducted within a building with no line-of-sight and no obstacles. Therefore, we used a path loss exponent  $n = 2.8$  [LRD07]. Using the described method for estimating  $A$ , we identified  $A = 58.6$  as a suitable value.

This setup is shown in Figure 7.6. The yellow point denotes the AP, while the green points illustrate the mobile phones. The target that should be localized is visualized by a red point. The center Nexus 5 is the initiating node that requests the measured distances from all other devices. Then, the center node computes the position of the victim.

Performing trilateration on the Wi-Fi chip consumes 250 mW power. If all frames were transferred to user space, power consumption would increase up to 565 mW on the average. Thus, the overall power used to locate and track victims is lower than if executed on the main CPUs. Furthermore, as shown in Figure 7.7, the power consumption of a passive device that only responds to trilateration requests with a list of distances is about 350 mW on the average. The

trilaterating node has an average power consumption of 390 mW. At 1.2 W for the active node and 800 mW for the passive node after about 25 seconds, a successful trilateration is indicated, where all nodes propagate the location of the victim. This consumes additional power due to the process that is required to build a custom Wi-Fi and IP packet with localization information. We repeated the experiment several times, while the Nexus 5 with the power monitor was set as the active part in half of the tests and to the passive part in the other half. Averaging all tests leads to the results shown in Figure 7.7.

### 7.7 Summary

This chapter provided the novel idea of transitional near-\* computing for emergency response applications. The challenges for such applications were identified as network overload in the remaining parts of the IP infrastructure, battery-efficiency and disruption tolerance.

These challenges were addressed by the following solutions based on transitional computing:

- Dynamically placed resource-aware gateways apply two kinds of computational transitions. In order to transform unaffected multi-homed mobile phones with a working Wi-Fi or cellular network connection into mobile gateways, network transitions (NT) are performed in combination with transitions between network service locations (CT 1).
- Opportunistic Named Functions (ONFs) in an Information-Centric Disruption-Tolerant Network (DTN) performed transitions between alternative opportunistic named functions (CT 2) to execute functions on-device. The selection between alternative functions is done according to the device's context.
- Battery-efficient context sensing in mobile devices was sensed battery-efficiently on-device. To achieve this, transitions between modes of context sensors and in-door localization protocols (CT 2) were performed.

The design and implementation of these solution were presented and experimental results were shown.

# 8

## Conclusions

This chapter concludes the thesis. First, the contributions of this thesis are summarized (Section 8.1). Then, areas of future work for transitional computing are outlined (Section 8.2).

### 8.1 Summary

In this thesis, the higher-level objective of increasing the energy efficiency of near-\* computing environments was addressed.

The novel term of near-\* computing was introduced. It combines four trends focusing on locality of data processing and resource provisioning: near-user, near-network, near-device, and near-data. Combined, they form a novel perspective on current computing paradigms summarized as near-\* computing.

Furthermore, the novel paradigm of transitional computing was defined. It includes three novel computational transitions:

- Transitions between locations (CT 1) are used to relocate functions and services to other physical or virtual hosts in a network.
- Transitions between implementations (CT 2) refer to the exchange of the implementations at runtime.
- Transitions between modes (CT 3) describe transitions between functions/services in different software or hardware layers (modes) of a computer system.

For each of these types of computational transitions, two realizations were presented.

First, two novel realizations of transitions between locations were introduced:

- Virtual machine consolidation realize transitions between VM locations (CT 1) in IaaS clouds. VM live migrations are performed for VM consolidation, allowing more servers in a cluster to power down.
- Dynamically assigned roles realize transitions between service locations (CT 1) within software-defined wireless networks. Furthermore, in order to save battery power of the involved devices, the arrangement of application and network services was combined with transitions between network mechanisms (NT).

Both realizations allow computing infrastructures to apply dynamic service migration and role assignments in order to adapt to user mobility and demand variations. In addition, they reduce the power consumption of private IaaS clouds and devices in a software-defined wireless network considerably.

Furthermore, two novel realizations of transitions between implementations were introduced:

- Data sparsing for MapReduce leverages transitions between input stream operators at the file system level (CT 2).
- ONFs realize transitions between named functions (CT 2) within ICN-DTNs. This allows opportunistic adaptations, i.e., the adaptation to locally available network and battery resources.

These realizations reduced the power consumption of MapReduce processing clusters and saved network bandwidth in an information-centric delay-tolerant network (ICN-DTN) considerably.

Finally, two novel realizations of transitions between modes were introduced:

- Multimodal CEP realizes transitions between modes of CEP operators (CT 3). It automatically breaks up CEP queries and selects the most adequate execution mode for the involved CEP operators. Filter, aggregation, and correlation operators can be expressed in a high-level language without requiring system-level domain-specific knowledge.
- Reactive programming of Wi-Fi firmware on mobile devices realizes transitions between modes of reactives (CT 3). The execution of reactive programs on the Wi-Fi chip, instead of running them on the CPU, is particularly useful for resource-constrained, low-latency applications.

These realizations reduced power consumption and improved latency of several network and application services considerably.

Finally, the novel idea of transitional near-\* computing for emergency response applications was presented. The challenges for such applications were identified as network overload in the remaining parts of the IP infrastructure, battery-efficiency, and disruption tolerance. These challenges were addressed by solutions based on transitional computing. Computational transitions enable an adaptive emergency response application, even if the Internet and connections to cloud services may be disrupted.

## 8.2 Future Work

Several areas of future work were already addressed for transitions between locations (CT 1), implementations (CT 2), and modes (CT 3) in the previous chapters. Additionally, there are several areas of future work to further advance transitional near-\* computing as a whole. They are discussed below.

### Virtualization in Near-Data Processing

Recent proposals like accelerator-centric operating systems [Sil17] make use of a current trend in high-end I/O-devices: fixed-function processors like network, I/O, and GPU accelerators alongside FPGAs provide interfaces for general purpose computations. They already allow skillful developers to program near-storage or near-network data processing features without involvement of the main CPU [Do+13; Sil17].

These fixed-function processors and accelerators could also be used to reduce latencies and power consumption in transitional near-\* computing. Unfortunately, this raises security and reliability issues. To address them, lightweight virtualization environments for near-storage or near-network processing on hardware accelerators need to be developed. One candidate for such a virtualization approach stems from the domain of virtualized network functions: BSD packet filters (BPF). In recent years, this approach has been advanced from a register-based evaluator [MJ93] to a universal in-kernel virtual machine used for network traffic control, firewalls and tracing. It is also capable of communicating with user-level processes (extended or eBPF<sup>1</sup>). Furthermore, eBPF addresses both reliability and security of a system. For reliability reasons, an in-kernel verifier statically determines that the eBPF does not exceed certain limits, such as a maximum number of instructions. And for security reasons, only a permitted set of functions can be called from within a BPF program. Introducing similar lightweight virtualization on hardware accelerators for near-storage or near-network processing in transitional near-\* computing seems to be a promising approach for future work.

### Transitional Serverless Computing

Serverless computing is a new cloud and fog computing paradigm where an application consists of individual pieces of application logic (functions) that can be separately managed and executed. One benefit of serverless computing is that it allows applications and services to be run without considering infrastructure requirements, such as provisioning, scaling, and managing. Examples of serverless computing frameworks are OpenLambda [Hen+16], OpenFaaS<sup>2</sup>, and Apache OpenWhisk<sup>3</sup>. They make use of sandboxed execution environments for pre-packaged services to deploy and run individual functions. Typically, these solutions implement a distributed, event-driven architecture on top of other infrastructure services like execution environments, storage, and databases. They vary in how individual functions are deployed, called, chained, and balanced [Akk+18].

---

<sup>1</sup><http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/networking/filter.txt>

<sup>2</sup><http://www.openfaas.com>

<sup>3</sup><http://openwhisk.apache.org>

These properties and features make serverless computing a perfect candidate for the application of transitional near-\* computing. Transitions between function implementations and between different topologies of chained functions could be applied: functions could be either totally or partially executed on mobile devices, network components, or within the cloud in order to improve their response time and to reduce their power consumption.

### **Transitional Computing for Machine Learning**

Binary neural networks (BNNs) [KS16; CBD15] have recently been used in resource-constrained environments like embedded devices. Similar to other approaches such as Approximate Convolutional Neural Networks [Cin+18], binary neural networks try to minimize the computational complexity of a trained artificial neural network by replacing operations with approximations. BNNs are employed to reduce the memory footprint and computational complexity using compression: floating point operations are replaced by bitwise logic. Exemplary use cases for BNNs have been deployed on embedded devices for classifying hand-written digits, for denoising human speech [KS18], for classifying phonemes, and for detecting human activities [EK16].

BNNs are a promising approach for adaptive machine learning applications. Within the paradigm of transitional computing, transitions between BNNs and full-fledged neural networks could be used to dynamically adapt machine learning applications to currently available resources like embedded Internet-connected devices, within the network, or on the lower layers of a computing system.

# Appendix



# List of Figures

<b>Chapter 2: Fundamentals</b>	<b>7</b>
2.1	OSI Reference Model . . . . . 8
2.2	SDN Architecture . . . . . 9
2.3	IEEE 802 Family and its Relation to the OSI Model . . . . . 10
2.4	Software-Defined Wireless Networking Stack (Linux) . . . . . 11
2.5	Network Function Virtualization (NFV) . . . . . 14
2.6	Fog Computing Overview . . . . . 17
2.7	Mobile Cloud Computing Overview . . . . . 18
2.8	Mobile Edge Computing Overview . . . . . 19
2.9	Task Migration within Asymmetric Multiprocessors . . . . . 22
2.10	Overview over Smartphone Subsystems . . . . . 24
<b>Chapter 3: Transitional Near-* Computing</b>	<b>27</b>
3.1	Multi-Mechanisms in a Communication System . . . . . 30
3.2	Multi-Mechanisms Adaptation Accross Layers . . . . . 31
3.3	Computational Transitions . . . . . 34
3.4	Transitions in Transitional Near-* Computing . . . . . 35
<b>Chapter 4: Transitions between Locations in Near-* Computing</b>	<b>39</b>
4.1	Virtual Machine Consolidation in IaaS-Clouds . . . . . 40
4.2	Dynamic Role Assignment in SDWNs . . . . . 40
<b>Section 4.2: Virtual Machine Consolidation in IaaS-Clouds</b>	<b>41</b>
4.2.1	Computational Transitions within VM Consolidation in IaaS-Clouds . . 41
4.2.2	Overview of the Eucalyptus Architecture . . . . . 42
4.2.3	Temporal Course of a VM Relocation Process . . . . . 46
4.2.4	Scheme of Eucalyptus Instance Images . . . . . 47
4.2.5	Scheme of a Modified Eucalyptus Instance . . . . . 48
4.2.6	Startup/Terminate Experimental Results . . . . . 50
4.2.7	MapReduce Wordcount Experimental Results . . . . . 51
4.2.8	Wordcount Energy Consumption Trend . . . . . 53
4.2.9	MapReduce PiEstimation Experimental Results . . . . . 54
<b>Section 4.3: Dynamic Role Assignment in SDWNs</b>	<b>58</b>

4.3.1	Computational Transitions within Dynamic Role Assignment in SDWNs	58
4.3.2	A Network Using Dynamic Role Assignment . . . . .	59
4.3.3	Experimental Setups . . . . .	63
4.3.4	Idle Power Consumption . . . . .	64
4.3.5	Power Consumption Under Load (Chained AP) . . . . .	65
4.3.6	Bandwidth Measurements . . . . .	66
4.3.7	Round Trip Times . . . . .	67
4.3.8	Power Consumption During Caching and Role Transition . . . . .	68
 <b>Chapter 5: Transitions between Implementations in Near-* Computing</b>		<b>73</b>
5.1	Data Sparsing for MapReduce . . . . .	74
5.2	Opportunistic Named Functions . . . . .	74
 <b>Section 5.2: Data Sparsing for MapReduce</b>		<b>75</b>
5.2.1	Computational Transitions within Data Sparsing for MapReduce . . . . .	75
5.2.2	The Principle of Sparsed Data Processing. . . . .	78
5.2.3	A Hadoop Cluster Including Data Stream Operators at the Local File System Layer . . . . .	79
5.2.4	Implementation of Transitions Between Data Stream Operators . . . . .	80
5.2.5	Specification of Operations to be Performed on Block Level . . . . .	81
5.2.6	Face Detection (FD) and Face Recognition (FR) Results . . . . .	84
5.2.7	Face Detection and Face Recognition Energy Consumption Results . . . . .	85
5.2.8	Speech Recognition Results . . . . .	86
5.2.9	Speech Recognition Energy Consumption . . . . .	87
 <b>Section 5.3: Opportunistic Named Functions</b>		<b>90</b>
5.3.1	Computational Transitions within Opportunistic Named Functions . . . . .	90
5.3.2	ONF Basic Concept . . . . .	91
5.3.3	Processing Named Content with ONFs . . . . .	92
5.3.4	ONF Functionality . . . . .	92
5.3.5	Regions of Interest for Relevant Topics . . . . .	97
5.3.6	Energy Consumed for a Face Detection-ONF . . . . .	99
5.3.7	Disaster Scenario . . . . .	100
 <b>Chapter 6: Transitions between Modes in Near-* Computing</b>		<b>107</b>
6.1	Multimodal Complex Event Processing . . . . .	108
6.2	Reactive Programming of Wi-Fi Firmware . . . . .	108
 <b>Section 6.2: Multimodal Complex Event Processing</b>		<b>109</b>
6.2.1	Computational Transitions within Multimodal CEP . . . . .	109
6.2.2	CEP Operators in Different Modes . . . . .	111
6.2.3	Operator Tree of CQL Query Examples . . . . .	113
6.2.4	Multimodal CEP Architecture in Android . . . . .	116

6.2.5	Quality-of-Experience (QoE) Execution Model and Code Generation . . .	118
6.2.6	Power Consumption of CEP Operators: User/Kernel Mode . . . . .	123
6.2.7	Power Consumption of CEP Operators: Wi-Fi/Kernel Mode . . . . .	124
6.2.8	Power Consumption of CEP Operators on the Microcontroller . . . . .	126
6.2.9	Use Case: Power Consumption . . . . .	128
<b>Section 6.3: Reactive Programming of Wi-Fi Firmware on Mobile Devices</b>		<b>131</b>
6.3.1	Computational Transitions within Reactive Programming of Wi-Fi Firm- wares on Mobile Devices . . . . .	131
6.3.2	Compiling an Reactive Program . . . . .	134
6.3.3	Extensions of Nexmon . . . . .	135
6.3.4	Reactive Program Definition . . . . .	136
6.3.5	Scheduling Reactives . . . . .	137
6.3.6	Nearby Video Streaming: Experimental Setup . . . . .	140
6.3.7	Nearby Video Streaming: Throughput and Video Quality . . . . .	141
6.3.8	Nearby Video Streaming: Power Consumption Baselines . . . . .	143
6.3.9	Nearby Video Streaming: Power Consumption . . . . .	143
6.3.10	Virtual Network Stack: Power Consumption Baselines . . . . .	146
6.3.11	Virtual Network Stack: Power Consumption . . . . .	147
6.3.12	Virtual Network Stack: Round Trip Times (RTTs) . . . . .	148
<b>Chapter 7: Transitional Near-* Computing for Emergency Response Applica- tions</b>		<b>153</b>
7.1	Transitional Near-* Computing for Emergency Response Applications . .	155
7.2	Transitions within an Adaptive Emergency Response Application . . .	157
7.3	Design of Dynamically Placed Bandwidth-Aware Gateways . . . . .	158
7.4	Search for Missing People in a Disaster Scenario . . . . .	162
7.5	Visualization of the Trilateration Approach . . . . .	165
7.6	In-Door Localization Experimental Setup . . . . .	167
7.7	In-Door Localization Experimental Results . . . . .	167



# List of Tables

2.1	Features of Different Computing Paradigms . . . . .	20
3.1	Contributions to Energy-efficient Transitional Near-* Computing. . . . .	36
4.2.1	ACPI Power Consumption . . . . .	44
4.3.1	Caching Setup Measurement Results . . . . .	67
5.2.1	Average File Size and Percentage of Modified FS Blocks. . . . .	83
5.2.2	Average File Size and Average Percentage of Modified FS Blocks . . . . .	86
5.3.1	Experimental Results . . . . .	101
6.2.1	Operator Algebra for Event Streams . . . . .	110
6.2.2	Calculation of Output Event Rate Based on Defined Windows and Predicate Selectivity . . . . .	114
6.3.1	H.264 Encoded Video Configurations for Dynamic Adaptive Streaming over HTTP (DASH) . . . . .	140
7.1	IPS Setup Measurement Results . . . . .	159
7.2	Search for Missing Persons: ONFs . . . . .	160
7.3	Search for Missing Persons: Interests . . . . .	161
7.4	RAM Usage of the Activity Detection Queries . . . . .	164



# Bibliography

- [AAS16] Zaafar Ahmed, Muhammad Hamad Alizai, and Affan A. Syed. “InKeV: In-Kernel Distributed Network Virtualization for DCN.” In: *SIGCOMM Computer Communication Review* 46.3. 2016, pp. 1–6 (page 129).
- [Abb+18] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. “Mobile Edge Computing: A Survey.” In: *IEEE Internet of Things Journal* 5.1. 2018, pp. 450–465 (page 19).
- [Abo+14] 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 and Tutorials* 16.1. 2014, pp. 337–368 (page 18).
- [ABS14] Carlos Anastasiades, Torsten Braun, and Vasilios A. Siris. “Information-Centric Networking in Mobile and Opportunistic Networks.” In: *Wireless Networking for Moving Objects - Protocols, Architectures, Tools, Services and Applications*. Vol. 8611. Lecture Notes in Computer Science. Springer, 2014, pp. 14–30 (page 102).
- [ABW06] Arvind Arasu, Shivnath Babu, and Jennifer Widom. “The CQL Continuous Query Language: Semantic Foundations and Query Execution.” In: *VLDB Journal* 15.2. 2006, pp. 121–142 (page 112).
- [Aga+09] Anant Agarwal, Martin Rinard, Stelios Sidiroglou, Sasa Misailovic, and Henry Hoffmann. “Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures”. Technical Report. MIT Dspace, 2009 (page 87).
- [AHO16] Mervat Abu-Elkheir, Hossam S. Hassanein, and Sharief M. A. Oteafy. “Enhancing Emergency Response Systems Through Leveraging Crowdsensing and Heterogeneous Data.” In: *Proceedings of the 12th International Wireless Communications and Mobile Computing Conference (IWCMC’16)*. IEEE, 2016, pp. 188–193 (page 73).
- [Ahr+08] Jeff Ahrenholz, Claudiu Danilov, Thomas R. Henderson, and Jae H. Kim. “CORE: A Real-time Network Emulator.” In: *Proceedings of the 27th IEEE Military Communications Conference (MILCOM 2008)*. IEEE, 2008, pp. 1–7 (page 99).
- [Aik+00] Bob Aiken, John Strassner, Brian E. Carpenter, Ian T. Foster, Clifford A. Lynch, Joe Mambretti, Reagan Moore, and Benjamin Teitelbaum. “Network Policy and Services: A Report of a Workshop on Middleware.” In: *RFC 2768*. 2000, pp. 1–29 (page 31).
- [Akb+15] Adnan Akbar, Francois Carrez, Klaus Moessner, Juan Sancho, and Juan Rico. “Context-Aware Stream Processing for Distributed IoT Applications.” In: *Proceedings of the 2nd IEEE World Forum on Internet of Things (WF-IoT 2015)*. 2015, pp. 663–668 (page 128).
- [Akk+18] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. “SAND: Towards High-Performance Serverless Computing.” In: *2018 USENIX Annual Technical Conference (ATC 2018)*. USENIX Association, 2018 (page 171).

- [Ant+14] Aleksandar Antonic, Kristijan Roankovic, Martina Marjanovic, Kreimir Pripuc, and Ivana Podnar Zarko. "A Mobile Crowdsensing Ecosystem Enabled by a Cloud-based Publish/Subscribe Middleware." In: *Proceedings of the 2nd International Conference on Future Internet of Things and Cloud (FiCloud 2014)*. IEEE, 2014, pp. 107–114 (page 128).
- [APO14] Kevin Abas, Caio Porto, and Katia Obraczka. "Wireless Smart Camera Networks for the Surveillance of Public Spaces." In: *IEEE Computer* 47.5. 2014, pp. 37–44 (page 103).
- [Arm+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. "A View of Cloud Computing." In: *Commun. ACM* 53.4. 2010, pp. 50–58 (page 44).
- [AV15] Syeda Persia Aziz and Nitin Vaidya. "Improving Reliability and Performance of Dense-AP Network Using DAPnet." In: *12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON 2015)*. IEEE, 2015, pp. 175–177 (page 68).
- [Bac+13] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt. "RIOT OS: Towards an OS for the Internet of Things." In: *Proceedings of the 32nd IEEE Conference on Computer Communications Workshops (INFOCOM 2013)*. 2013, pp. 79–80 (page 150).
- [Bau+10] Christian Baun, Marcel Kunze, Jens Nimis, and Stefan Tai. "Cloud Computing - Web-basierte dynamische IT-Services". Springer, Heidelberg, 2010 (pages 15, 16, 43).
- [Bau+12] Lars Baumgärtner, Pablo Graubner, Matthias Leinweber, Roland Schwarzkopf, Matthias Schmidt, Bernhard Seeger, and Bernd Freisleben. "Mastering Security Anomalies in Virtualized Computing Environments via Complex Event Processing." In: *Proceedings of the 4th International Conference on Information, Process, and Knowledge Management (eKNOW 2012)*. XPS, 2012, pp. 76–81.
- [Bau+15] Lars Baumgärtner, Pablo Graubner, Nils Schmidt, and Bernd Freisleben. "AndroLyze: A Distributed Framework for Efficient Android App Analysis." In: *Proceedings of the 4th IEEE International Conference on Mobile Services (MS 2015)*. IEEE, 2015, pp. 73–80.
- [Bau+16] Lars Baumgärtner, Paul Gardner-Stephen, Pablo Graubner, Jeremy Lakeman, Jonas Hochst, Patrick Lampe, Nils Schmidt, Stefan Schulz, Artur Sterz, and Bernd Freisleben. "An Experimental Evaluation of Delay-tolerant Networking with Serval." In: *Proceedings of the 6th IEEE Global Humanitarian Technology Conference (GHTC 2016)*. IEEE, 2016, pp. 70–79 (page 154).
- [Bau+17] 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: *Proceedings of the 13th Annual Conference on Wireless On-demand Network Systems and Services (WONS 2017)*. IEEE, 2017, pp. 33–40.
- [BB10a] Anton Beloglazov and Rajkumar Buyya. "Adaptive Threshold-Based Approach for Energy-Efficient Consolidation of Virtual Machines in Cloud Data Centers." In: *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science (MCG 2010)*. ACM, 2010, p. 4 (page 24).

- [BB10b] Anton Beloglazov and Rajkumar Buyya. “Energy Efficient Resource Management in Virtualized Cloud Data Centers.” In: *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, (CCGrid 2010)*. IEEE, 2010, pp. 826–831 (page 56).
- [BC10] Woongki Baek and Trishul M. Chilimbi. “Green: A Framework for Supporting Energy-conscious Programming Using Controlled Approximation.” In: *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2010)*. ACM, 2010, pp. 198–209 (pages 87, 88).
- [BCH13] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. “The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition”. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2013 (pages 23, 27, 77).
- [Bel+12] Paolo Bellavista, Antonio Corradi, Mario Fanelli, and Luca Foschini. “A Survey of Context Data Distribution for Mobile Ubiquitous Systems.” In: *ACM Comput. Surv.* 44.4. 2012, 24:1–24:45 (pages 31, 107, 110, 155, 156, 163).
- [Bel+17] Adam Belay, George Prekas, Mia Primorac, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. “The IX Operating System: Combining Low Latency, High Throughput, and Efficiency in a Protected Dataplane.” In: *ACM Trans. Comput. Syst.* 34.4. 2017, 11:1–11:39 (page 28).
- [Ber+14] C. J. Bernardos, A. de la Oliva, P. Serrano, A. Banchs, L. M. Contreras, H. Jin, and J. C. Zuniga. “An Architecture for Software Defined Wireless Networking.” In: *IEEE Wireless Communications* 21.3. 2014, pp. 52–61 (pages 69, 149).
- [BH07] Luiz André Barroso and Urs Hölzle. “The Case for Energy-proportional Computing.” In: *IEEE Computer* 40.12. 2007, pp. 33–37 (page 77).
- [Bia+12] G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, and I. Tinnirello. “MAClets: Active MAC Protocols over Hard-coded Devices.” In: *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2012)*. ACM, 2012, pp. 229–240 (page 149).
- [BM01] Ella Bingham and Heikki Mannila. “Random Projection in Dimensionality Reduction: Applications to Image and Text Data.” In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2001, pp. 245–250 (page 76).
- [Bon+12] Flavio Bonomi, Rodolfo A. Milito, Jiang Zhu, and Sateesh Addepalli. “Fog Computing and its Role in the Internet of Things.” In: *Proceedings of the 1st Edition of the MCC Workshop on Mobile Cloud Computing (MCC@SIGCOMM 2012)*. ACM, 2012, pp. 13–16 (page 27).
- [Bor+12] Damien Borgetto, Michael Maurer, Georges Da Costa, Jean-Marc Pierson, and Ivona Brandic. “Energy-Efficient and SLA-Aware Management of IaaS Clouds.” In: *Proceedings of the 3rd International Conference on Energy-Efficient Computing and Networking (e-Energy’12)*. ACM, 2012, p. 25 (page 24).
- [Bot+09] Irina Botan, Younggoo Cho, Roozbeh Derakhshan, Nihal Dindar, Laura Haas, Kihong Kim, Chulwon Lee, Girish Mundada, M C Shan, and Nesime Tatbul. “Design and Implementation of the MaxStream Federated Stream Processing Architecture”. Technical Report. ETH Zurich, Department of Computer Science, 2009 (page 128).

- [Bou+10] Ghazi Bouabene, Christophe Jelger, Christian F. Tschudin, Stefan Schmid, Ariane Keller, and Martin May. “The Autonomic Network Architecture (ANA).” In: *IEEE Journal on Selected Areas in Communications* 28.1. 2010, pp. 4–14 (page 102).
- [BP00] P. Bahl and V. N. Padmanabhan. “RADAR: An In-Building RF-Based User Location and Tracking System.” In: *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)*. Vol. 2. 2000, 775–784 vol.2 (page 164).
- [BPT15] Mike Belshe, Roberto Peon, and Martin Thomson. “Hypertext Transfer Protocol Version 2 (HTTP/2).” In: *RFC 7540*. 2015, pp. 1–96 (page 8).
- [Bye+14] S. Byeon, K. Yoon, O. Lee, S. Choi, W. Cho, and S. Oh. “MoFA: Mobility-aware Frame Aggregation in Wi-Fi.” In: *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT 2014)*. ACM, 2014, pp. 41–52 (page 133).
- [Cal+10] Pat R. Calhoun, Rohit Suri, Nancy Cam-Winget, Michael G. Williams, Susan Hares, Bob O’Hara, and Scott G. Kelly. “Lightweight Access Point Protocol (LWAPP).” In: *RFC 5412*. 2010, pp. 1–125 (page 68).
- [CBD15] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. “BinaryConnect: Training Deep Neural Networks with Binary Weights During Propagations.” In: *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS 2015)*. 2015, pp. 3123–3131 (page 172).
- [CC12] Kefei Cheng and Yanglei Cui. “Design and Implementation of Network Packets Collection Tools Based on the Android Platform.” In: *Proceedings of the 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2012)*. IEEE, 2012, pp. 2166–2169 (page 129).
- [CG14] Marco Conti and Silvia Giordano. “Mobile Ad Hoc Networking: Milestones, Challenges, and New Research Directions.” In: *IEEE Communications Magazine* 52.1. 2014, pp. 85–96 (page 68).
- [CGK10] Yanpei Chen, Archana Ganapathi, and Randy H. Katz. “To Compress or not to Compress - Compute vs. I/O Tradeoffs for Mapreduce Energy Efficiency.” In: *Proceedings of the 1st ACM SIGCOMM Workshop on Green Networking*. ACM, 2010, pp. 23–28 (pages 77, 88).
- [Che+12] Yanpei Chen, Sara Alspaugh, Dhruva Borthakur, and Randy H. Katz. “Energy Efficiency for Large-scale MapReduce Workloads with Significant Interactive Analysis.” In: *Proceedings of the 7th EuroSys Conference (EuroSys 2012)*. ACM, 2012, pp. 43–56 (page 163).
- [Che+15] Jordan Cheney, Benjamin Klein, Anil K. Jain, and Brendan F. Klare. “Unconstrained Face Detection: State of the Art Baseline and Challenges.” In: *Proceedings of the International Conference on Biometrics (ICB 2015)*. IEEE, 2015, pp. 229–236 (pages 97, 98).
- [Che+16a] Jiachen Chen, Mayutan Arumaithurai, Xiaoming Fu, and K. K. Ramakrishnan. “CNS: Content-oriented Notification Service for Managing Disasters.” In: *Proceedings of the 3rd ACM Conference on Information-Centric Networking (ICN 2016)*. 2016, pp. 122–131 (page 103).

- [Che+16b] X. Chen, X. Gong, L. Yang, and J. Zhang. “Exploiting Social Tie Structure for Cooperative Wireless Networking: A Social Group Utility Maximization Framework.” In: *IEEE/ACM Transactions on Networking* 24.6. 2016, pp. 3593–3606 (page 132).
- [Che+17] C. Chen, H. Tong, L. Xie, L. Ying, and Q. He. “Cross-Dependency Inference in Multi-Layered Networks: A Collaborative Filtering Perspective.” In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 11.4. 2017, 42:1–42:26 (page 133).
- [Cin+18] R. J. Cintra, S. Duffner, C. Garcia, and A. Leite. “Low-Complexity Approximate Convolutional Neural Networks.” In: *IEEE Transactions on Neural Networks and Learning Systems*. 2018, pp. 1–12 (page 172).
- [Cis17] Cisco Systems, Inc. “Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update”. Technical Report. Cisco Systems, Inc., 2017 (page 138).
- [CKB14] Mathieu Cunche, Mohamed Ali Kâafar, and Rokhsana Boreli. “Linking Wireless Devices Using Information Contained in Wi-Fi Probe Requests.” In: *Pervasive and Mobile Computing* 11. 2014, pp. 56–69 (page 126).
- [Cla+05] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. “Live Migration of Virtual Machines.” In: *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI 2005)*. USENIX, 2005 (page 55).
- [CMS09] Pat R. Calhoun, Michael P. Montemurro, and Dorothy Stanley. “Control and Provisioning of Wireless Access Points (CAPWAP) Protocol Specification.” In: *RFC* 5415. 2009, pp. 1–155 (page 68).
- [Cos+12] Salvatore Costanzo, Laura Galluccio, Giacomo Morabito, and Sergio Palazzo. “Software Defined Wireless Networks: Unbridling SDNs.” In: *Proceedings of the 2012 European Workshop on Software Defined Networking (EWSDN 2012)*. IEEE, 2012, pp. 1–6 (page 149).
- [DA18] Tamer Dag and Taner Arsan. “Received Signal Strength Based Least Squares Lateration Algorithm for Indoor Localization.” In: *Computers & Electrical Engineering* 66. 2018, pp. 114–126 (page 164).
- [Das+17] R. Das, N. Baranasuriya, V. N. Padmanabhan, C. Rödbro, and S. Gilbert. “Informed Bandwidth Adaptation in Wi-Fi Networks Using Ping-Pair.” In: *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT 2017)*. ACM, 2017, pp. 376–388 (page 133).
- [DG03] Sanjoy Dasgupta and Anupam Gupta. “An Elementary Proof of a Theorem of Johnson and Lindenstrauss.” In: *Random Struct. Algorithms* 22.1. 2003, pp. 60–65 (page 76).
- [DGH14] L. Duan, L. Gao, and J. Huang. “Cooperative Spectrum Sharing: A Contract-Based Approach.” In: *IEEE Transactions on Mobile Computing (TMC)* 13.1. 2014, pp. 174–187 (page 132).
- [Do+13] Jaeyoung Do, Yang-Suk Kee, Jignesh M. Patel, Chanik Park, Kwanghyun Park, and David J. DeWitt. “Query Processing on Smart SSDs: Opportunities and Challenges.” In: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2013)*. ACM, 2013, pp. 1221–1230 (pages 28, 171).

- [Dua+12] L. Duan, T. Kubo, K. Sugiyama, J. Huang, T. Hasegawa, and J. Walrand. “Incentive Mechanisms for Smartphone Collaboration in Data Acquisition and Distributed Computing.” In: *Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM 2012)*. 2012, pp. 1701–1709 (page 133).
- [EK16] Marcus Edel and Enrico Koppe. “Binarized-BLSTM-RNN based Human Activity Recognition.” In: *Proceedings of the 2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN 2016)*. 2016, pp. 1–7 (page 172).
- [Esm+12] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. “Architecture Support for Disciplined Approximate Programming.” In: *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2012, London, UK, March 3-7, 2012*. ACM, 2012, pp. 301–312 (pages 87, 88).
- [Esm+15] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. “Neural Acceleration for General-purpose Approximate Programs.” In: *Commun. ACM* 58.1. 2015, pp. 105–115 (pages 78, 88).
- [Eva11] Dave Evans. “The Internet of Things How the Next Evolution of the Internet Is Changing Everything”. Technical Report 2011. 2011, pp. 1–11 (pages 1, 28).
- [Fon+17] R. Dos Reis Fontes, C. Campolo, C. Esteve Rothenberg, and A. Molinaro. “From Theory to Experimental Evaluation: Resource Management in Software-Defined Vehicular Networks.” In: *IEEE Access* 5. 2017, pp. 3069–3076 (page 149).
- [Fos+09] Ian T. Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. “Cloud Computing and Grid Computing 360-Degree Compared.” In: *CoRR* abs/0901.0131. 2009. arXiv: 0901.0131 (page 27).
- [Fou12a] ONF Networking Foundation. “OpenFlow Switch Specification.” In: *ONF White Paper*. 2012 (page 9).
- [Fou12b] ONF Networking Foundation. “Software-defined Networking: The New Norm for Networks.” In: *ONF White Paper*. 2012 (page 9).
- [Fre15] Julien Freudiger. “How Talkative is Your Mobile Device?: An Experimental Study of Wi-Fi Probe Requests.” In: *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec 2015)*. ACM, 2015, 8:1–8:6 (page 126).
- [Frö+15] Alexander Frömmgen, Björn Richerzhagen, Julius Rückert, David Hausheer, Ralf Steinmetz, and Alejandro P. Buchmann. “Towards the Description and Execution of Transitions in Networked Systems.” In: *Proceedings of the 9th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security (AIMS 2015)*. Vol. 9122. Lecture Notes in Computer Science. Springer, 2015, pp. 17–29 (page 29).
- [Frö+16] 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”. Darmstadt, 2016 (pages 3, 29–32).
- [Gar+12] 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 (page 95).

- [Gar+13a] 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 95, 96).
- [Gar+13b] 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: *Proceedings of the 3rd IEEE Global Humanitarian Technology Conference (GHTC 2013)*. IEEE, 2013, pp. 162–166 (page 95).
- [Gas05] Matthew S. Gast. “802.11 Wireless Networks - The Definitive Guide: Creating and Administering Wireless Networks (Second Edition)”. O’Reilly, 2005 (page 10).
- [Gay+14] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler. “The nesC Language: A Holistic Approach to Networked Embedded Systems.” In: *ACM Sigplan Notices* 49.4. 2014, pp. 41–51 (page 149).
- [GHF15] Pablo Graubner, Patrick Heckmann, and Bernd Freisleben. “Energy-Efficient Data Processing Through Data Sparsing with Artifacts.” In: *Proceedings of the 30th International Conference ISC High Performance (ISC HPC 2015)*. Vol. 9137. Lecture Notes in Computer Science. Springer, 2015, pp. 307–322 (pages xiii, 75).
- [Gia+] C. Giannini, P. Calegari, C. Buratti, and R. Verdone. “Delay Tolerant Network for Smart City: Exploiting Bus Mobility.” In: *Proceedings of the 2016 AEIT International Annual Conference (AEIT 2016)*, pp. 1–6 (page 103).
- [Gra+15] Pablo Graubner, Lars Baumgärtner, Patrick Heckmann, Marcel Müller, and Bernd Freisleben. “Dyalize: Dynamic Analysis of Mobile Apps in a Platform-as-a-Service Cloud.” In: *Proceedings of the 8th IEEE International Conference on Cloud Computing (CLOUD 2015)*. IEEE, 2015, pp. 925–932.
- [Gra+17] Pablo Graubner, Markus Sommer, Matthias Hollick, and Bernd Freisleben. “Dynamic Role Assignment in Software-Defined Wireless Networks.” In: *Proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC 2017)*. IEEE, 2017, pp. 760–766 (pages xiii, xiv, 58, 153).
- [Gra+18a] Pablo Graubner, Patrick Lampe, Jonas Hochst, Lars Baumgärtner, Mira Mezini, and Bernd Freisleben. “Opportunistic Named Functions in Disruption-tolerant Emergency Networks.” In: *Proceedings of the 15th ACM International Conference on Computing Frontiers (CF 2018)*. ACM, 2018, pp. 129–137 (pages xiii, xiv, 90, 153).
- [Gra+18b] 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 ACM International Conference on Distributed and Event-based Systems (DEBS 2018)*. ACM, 2018, pp. 112–123 (pages xiii, xiv, 109, 153).
- [Gro17] OpenFog Consortium Architecture Working Group. “OpenFog Reference Architecture for Fog Computing”. OpenFog Consortium White Paper. 2017 (pages 16, 17).
- [GSF11] Pablo Graubner, Matthias Schmidt, and Bernd Freisleben. “Energy-Efficient Management of Virtual Machines in Eucalyptus.” In: *Proceedings of the 4th IEEE International Conference on Cloud Computing (CLOUD 2011)*. IEEE, 2011, pp. 243–250 (pages xiii, 41).

- [GSF13] Pablo Graubner, Matthias Schmidt, and Bernd Freisleben. “Energy-Efficient Virtual Machine Consolidation.” In: *IT Professional* 15.2. 2013, pp. 28–34 (pages xiii, 41).
- [Has+11] Katharina Haselhorst, Matthias Schmidt, Roland Schwarzkopf, Niels Fallenbeck, and Bernd Freisleben. “Efficient Storage Synchronization for Live Migration in Cloud Infrastructures.” In: *Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2011)*. 2011, pp. 511–518 (page 45).
- [Hät+16] Seppo Hätönen, Petri Savolainen, Ashwin Rao, Hannu Flinck, and Sasu Tarkoma. “Off-the-Shelf Software-defined Wi-Fi Networks.” In: *Proceedings of the ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 609–610 (page 149).
- [HBC17] George Haddow, Jane Bullock, and Damon P Coppola. “Introduction to Emergency Management”. Butterworth-Heinemann, 2017 (page 153).
- [Hed+14] Ward Van Heddeghem, Sofie Lambert, Bart Lannoo, Didier Colle, Mario Pickavet, and Piet Demeester. “Trends in Worldwide ICT Electricity Consumption from 2007 to 2012.” In: *Computer Communications* 50. 2014, pp. 64–76 (page 2).
- [Hen+16] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. “Serverless Computation with OpenLambda.” In: *Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2016)*. USENIX Association, 2016 (page 171).
- [HK09] I. Hou and P. R. Kumar. “Admission Control and Scheduling for QoS Guarantees for Variable-Bit-Rate Applications on Wireless Channels.” In: *Proceedings of the 10th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2009)*. 2009, pp. 175–184 (page 132).
- [HLC10] Ting-Kai Huang, Chia-Keng Lee, and Ling-Jyh Chen. “PROPHET+: An Adaptive PROPHET-Based Routing Protocol for Opportunistic Network.” In: *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010)*. IEEE, 2010, pp. 112–119 (page 103).
- [HLL16] M. S. Ul Haq, L. Liao, and M. Lerong. “Design and Implementation of a Sandbox Technique for Isolated Applications.” In: *Proceedings of the 1st IEEE Information Technology, Networking, Electronic and Automation Control Conference (IAEAC 2016)*. 2016, pp. 557–561 (page 129).
- [Hoa+13] Dinh Thai Hoang, Chonho Lee, Dusit Niyato, and Ping Wang. “A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches.” In: *Wireless Communications and Mobile Computing* 13.18. 2013, pp. 1587–1611 (page 27).
- [Hos+16] Jan Hendrik Hosang, Rodrigo Benenson, Piotr Dollár, and Bernt Schiele. “What Makes for Effective Detection Proposals?” In: *IEEE Trans. Pattern Anal. Mach. Intell.* 38.4. 2016, pp. 814–830 (page 97).
- [Hou14] I. Hou. “Scheduling Heterogeneous Real-Time Traffic over Fading Wireless Channels.” In: *Transactions on Networking (TON)* 22.5. 2014, pp. 1631–1644 (page 132).
- [Hu+15] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. “Mobile Edge Computing - A Key Technology Towards 5G”. ETSI White Paper 11. 2015, pp. 1–16 (pages 14, 19, 20).

- [Ior+18] Michaela Iorga, Larry Feldman, Robert Barton, Michael J. Martin, Nedim S. Goren, and Charif Mahmoudi. “NIST Special Publication 500-325: Fog Computing Conceptual Model”. NIST Special Publication. 2018 (pages 17, 20, 28).
- [KB10] Rini T. Kaushik and Milind Bhandarkar. “GreenHDFS: Towards an Energy-conserving, Storage-efficient, Hybrid Hadoop Compute Cluster.” In: *Proceedings of the 2010 International Conference on Power Aware Computing and Systems (HotPower 2010)*. Vancouver, BC, Canada: USENIX Association, 2010, pp. 1–9 (page 88).
- [KF13] H. Kim and N. Feamster. “Improving Network Management with Software Defined Networking.” In: *IEEE Communications Magazine* 51.2. 2013, pp. 114–119 (page 149).
- [KH18] Joo-Ho Kim and Makarand Hastak. “Social Network Analysis: Characteristics of Online Social Networks After a Disaster.” In: *Int J. Information Management* 38.1. 2018, pp. 86–96 (page 158).
- [Kil+13] Haim Kilov, Peter F. Linington, José Raúl Romero, Akira Tanaka, and Antonio Vallecillo. “The Reference Model of Open Distributed Processing: Foundations, Experience and Applications.” In: *Computer Standards & Interfaces* 35.3. 2013, pp. 247–256 (page 32).
- [Kim+15] Suhwuk Kim, Yuki Urata, Yuki Koizumi, and Toru Hasegawa. “Power-saving NDN-based Message Delivery based on Collaborative Communication in Disasters.” In: *Proceedings of the 21st IEEE International Workshop on Local and Metropolitan Area Networks (LANMAN 2015)*. IEEE, 2015, pp. 1–6 (page 103).
- [Kjæ+09] Mikkel Baun Kjærgaard, Jakob Langdal, Torben Godsk, and Thomas Toftkjær. “EnTracked: Energy-Efficient Robust Position Tracking for Mobile Devices.” In: *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services (MobiSys 2009)*. ACM, 2009, pp. 221–234 (page 128).
- [KKC18] Y. G. Kim, J. Kong, and S. W. Chung. “A Survey on Recent OS-Level Energy Management Techniques for Mobile Processing Units.” In: *IEEE Transactions on Parallel and Distributed Systems* 29.10. 2018, pp. 2388–2401 (page 24).
- [Kre+15] Diego Kreutz, Fernando M. V. Ramos, Paulo Jorge Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. “Software-Defined Networking: A Comprehensive Survey.” In: *Proceedings of the IEEE* 103.1. 2015, pp. 14–76 (page 149).
- [KS09] 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 (page 110).
- [KS16] Minje Kim and Paris Smaragdis. “Bitwise Neural Networks.” In: *CoRR*. 2016. arXiv: 1601.06071 (page 172).
- [KS18] Minje Kim and Paris Smaragdis. “Bitwise Neural Networks for Efficient Single-Channel Source Separation.” In: *Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2018)*. 2018, pp. 701–705 (page 172).
- [Ku+14] I. Ku, Y. Lu, M. Gerla, R. L. Gomes, F. Ongaro, and E. Cerqueira. “Towards Software-Defined VANET: Architecture and Services.” In: *Proceedings of the 13th Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET 2014)*. 2014, pp. 103–110 (page 149).

- [Lam+17] Patrick Lampe, Lars Baumgärtner, Ralf Steinmetz, and Bernd Freisleben. “Smart-Face: Efficient Face Detection on Smartphones for Wireless On-demand Emergency Networks.” In: *Proceedings of the 24th International Conference on Telecommunications (ICT 2017)*. IEEE, 2017, pp. 1–7 (pages 98, 160).
- [Lee+14] Jeongkeun Lee, Mostafa Uddin, Jean Tourrilhes, Souvik Sen, Sujata Banerjee, Manfred Arndt, Kyu-Han Kim, and Tamer Nadeem. “meSDN: Mobile Extension of SDN.” In: *Proceedings of the Fifth International Workshop on Mobile Cloud Computing & Services (MCS 2014)*. ACM, 2014, pp. 7–14 (page 69).
- [Lev+05] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. “TinyOS: An Operating System for Sensor Networks.” In: *Ambient Intelligence*. Springer Berlin Heidelberg, 2005, pp. 115–148 (pages 128, 149).
- [LHC06] Ping Li, Trevor Hastie, and Kenneth Ward Church. “Very Sparse Random Projections.” In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2006, pp. 287–296 (page 76).
- [Li+17] Y. Li, J. Gao, P. P. C. Lee, L. Su, C. He, C. He, F. Yang, and W. Fan. “A Weighted Crowdsourcing Approach for Network Quality Measurement in Cellular Data Networks.” In: *IEEE Transactions on Mobile Computing (TMC)* 16.2. 2017, pp. 300–313 (page 133).
- [Li+18] Chao Li, Yushu Xue, Jing Wang, Weigong Zhang, and Tao Li. “Edge-Oriented Computing Paradigms: A Survey on Architecture Design and System Management.” In: *ACM Computing Surveys* 51.2. 2018, 39:1–39:34 (page 2).
- [Lia+16] Daniyal Liaqat, Silviu Jingoi, Eyal de Lara, Ashvin Goel, Wilson To, Kevin Lee, Italo De Moraes Garcia, and Manuel Saldaña. “Sidewinder: An Energy Efficient and Developer Friendly Heterogeneous Architecture for Continuous Mobile Sensing.” In: *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2016)*. ACM, 2016, pp. 205–215 (page 129).
- [LJJ17] Hyo Won Lee, Wha Sook Jeon, and Dong Geun Jeong. “A Practical Indoor Localization Scheme for Disaster Relief.” In: *Proceedings of the 85th IEEE Vehicular Technology Conference, (VTC 2017-Spring)*. IEEE, 2017, pp. 1–7 (page 163).
- [LLB15] Matthew Lentz, James Litton, and Bobby Bhattacharjee. “Drowsy Power Management.” In: *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP 2015)*. ACM, 2015, pp. 230–244 (page 129).
- [LP17] Sofie Lambert and Mario Pickavet. “Can the Internet Be Greener?” In: *Proceedings of the IEEE* 105.2. 2017, pp. 179–182 (page 2).
- [LRD07] Lorne Liechty, Eric Reifsnider, and Greg Durgin. “Developing the Best 2.4 GHz Propagation Model from Active Network Measurements.” In: *Proceedings of the 66th IEEE Vehicular Technology Conference (VTC 2007-Fall)*. IEEE, 2007, pp. 894–896 (page 167).
- [Lu+10] 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: *Proceedings of the 8th International Conference on Embedded Networked Sensor Systems (SenSys 2010)*. ACM, 2010, pp. 71–84 (page 128).

- [LZ15] Ping Li and Cun-Hui Zhang. “Compressed Sensing with Very Sparse Gaussian Random Projections.” In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2015)*. Vol. 38. JMLR Workshop and Conference Proceedings. JMLR.org, 2015 (page 76).
- [Mad+05] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. “Tiny-DB: An Acquisitional Query Processing System for Sensor Networks.” In: *ACM Trans. Database Syst.* 30.1. 2005, pp. 122–173 (page 128).
- [Mar+16] M. Marjanović, L. Skorin-Kapov, K. Pripuzić, A. Antonić, and I. Podnar Žarko. “Energy-aware and Quality-driven Sensor Management for Green Mobile Crowd Sensing.” In: *J. Netw. Comput. Appl.* 59.C. 2016, pp. 95–108 (page 128).
- [Men+15] Flavio Meneses, Daniel Corujo, Carlos Guimarães, and Rui L. Aguiar. “Extending SDN to End Nodes Towards Heterogeneous Wireless Mobility.” In: *2015 IEEE Global Communications Conference (Globecom 2015) Workshops*. IEEE, 2015, pp. 1–6 (page 69).
- [Mic14] Arthur Mickoleit. “Social Media Use by Governments: A Policy Primer to Discuss Trends, Identify Policy Opportunities and Guide Decision MakersNo.” In: *OECD Working Papers on Public Governance* 26. 2014 (page 158).
- [Mil+08] Emiliano Miluzzo, Nicholas D. Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B. Eisenman, Xiao Zheng, and Andrew T. Campbell. “Sensing Neets Mobile Social Networks: The Design, Implementation and Evaluation of the Cenceme Application.” In: *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys 2008)*. ACM, 2008, pp. 337–350 (page 128).
- [Mit16] Sparsh Mittal. “A Survey of Techniques for Architecting and Managing Asymmetric Multicore Processors.” In: *ACM Comput. Surv.* 48.3. 2016, 45:1–45:38 (pages 22, 23).
- [MJ93] Steven McCanne and Van Jacobson. “The BSD Packet Filter: A New Architecture for User-level Packet Capture.” In: *Proceedings of the Usenix Winter 1993 Technical Conference*. USENIX Association, 1993, pp. 259–270 (page 171).
- [MKC12] Radhika Mittal, Aman Kansal, and Ranveer Chandra. “Empowering Developers to Estimate App Energy Consumption.” In: *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (Mobicom 2012)*. ACM, 2012, pp. 317–328 (page 25).
- [MLP16] Lenord Melvix J. S. M., Vikas Lokesh, and George C. Polyzos. “Energy Efficient Context Based Forwarding Strategy in Named Data Networking of Things.” In: *Proceedings of the 3rd ACM Conference on Information-Centric Networking (ICN 2016)*. 2016, pp. 249–254 (page 104).
- [MNR13] Prashanth Mohan, Suman Nath, and Oriana Riva. “Prefetching Mobile Ads: Can Advertising Systems Afford It?” In: *Proceedings of the 8th Eurosys Conference (EuroSys 2013)*. ACM, 2013, pp. 267–280 (page 25).
- [MNW14] Marshall Kirk McKusick, George V Neville-Neil, and Robert NM Watson. “The Design and Implementation of the FreeBSD Operating System”. Pearson Education, 2014 (page 13).

- [Mon+14] Edo Monticelli, Benno M. Schubert, Mayutan Arumaithurai, Xiaoming Fu, and K. K. Ramakrishnan. “An Information Centric Approach for Communications in Disaster Situations.” In: *Proceedings of the 20th IEEE International Workshop on Local & Metropolitan Area Networks (LANMAN 2014)*. 2014, pp. 1–6 (pages 96, 103).
- [Moo65] Gordon E. Moore. “Cramming More Components onto Integrated Circuits.” In: *Electronics* 38.8. 1965, pp. 114–117 (page 1).
- [MPZ10] Michael Meisel, Vasileios Pappas, and Lixia Zhang. “Listen First, Broadcast Later: Topology-agnostic Forwarding under High Dynamics.” In: *Annual Conference of the Int. Technology Alliance in Network and Information Science*. 2010, p. 8 (page 102).
- [MV13] Vijay Mahadevan and Nuno Vasconcelos. “On the Plausibility of the Discriminant Center-surround Hypothesis for Visual Saliency.” In: *IEEE Trans. Pattern Anal. Mach. Intell.* 35.3. 2013, pp. 541–554 (page 97).
- [Nai15] Ravi Nair. “Big Data Needs Approximate Computing: Technical Perspective.” In: *Commun. ACM* 58.1. 2015, p. 104 (page 78).
- [Ngu+17] The An Binh Nguyen, Pratyush Agnihotri, Christian Meurisch, Manisha Luthra, Rahul Chini Dwarakanath, Jeremias Blendin, Doreen Böhnstedt, Michael Zink, and Ralf Steinmetz. “Efficient Crowd Sensing Task Distribution Through Context-aware NDN-based Geocast.” In: *Proceedings of the 42nd IEEE Conference on Local Computer Networks (LCN 2017)*. IEEE, 2017, pp. 52–60 (page 104).
- [NS07] Ripal Nathuji and Karsten Schwan. “VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems.” In: *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP 2007)*. ACM, 2007, pp. 265–278 (page 56).
- [Nun+14] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turetli. “A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks.” In: *IEEE Communications Surveys Tutorials* 16.3. 2014, pp. 1617–1634 (page 149).
- [Nur+08] D. Nurmi, R. Wolski, C. Grzegorzcyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. “Eucalyptus : A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems”. Technical Report. Computer Science Department, University of California, 2008 (page 43).
- [Nur+09] Daniel Nurmi, Richard Wolski, Chris Grzegorzcyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. “The Eucalyptus Open-Source Cloud-Computing System.” In: *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, (CCGrid 2009)*. IEEE, 2009, pp. 124–131 (pages 41, 42).
- [OBL15] Gabriel Orsini, Dirk Bade, and Winfried Lamersdorf. “Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading.” In: *Proceedings of the 8th IFIP Wireless and Mobile Networking Conference (WMNC 2015)*. IEEE, 2015, pp. 112–119 (page 27).
- [OLG10] Soon-Young Oh, Davide Lau, and Mario Gerla. “Content Centric Networking in Tactical and Emergency MANETs.” In: *Proceedings of the 3rd IFIP Wireless Days Conference 2010*. IEEE, 2010, pp. 1–5 (pages 102, 154).
- [Par66] Douglas F. Parkhill. “The Challenge of the Computer Utility”. 1966 (page 15).

- [Pet+16] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas E. Anderson, and Timothy Roscoe. “Arrakis: The Operating System Is the Control Plane.” In: *ACM Trans. Comput. Syst.* 33.4. 2016, 11:1–11:30 (page 28).
- [PG74] Gerald J. Popek and Robert P. Goldberg. “Formal Requirements for Virtualizable Third Generation Architectures.” In: *Commun. ACM* 17.7. 1974, pp. 412–421 (pages 12, 13).
- [PH15] Marcus Pinnecke and Bastian Hoßbach. “Query Optimization in Heterogenous Event Processing Federations.” In: *Datenbank Spektrum* 15.3. 2015, pp. 193–202 (page 128).
- [Pos80] Jon Postel. “User Datagram Protocol.” In: *RFC* 768. 1980, pp. 1–3 (page 8).
- [Pos81] Jon Postel. “Transmission Control Protocol.” In: *RFC* 793. 1981, pp. 1–91 (page 8).
- [Psa+14] Ioannis Psaras, Lorenzo Saino, Mayutan Arumaithurai, K. K. Ramakrishnan, and George Pavlou. “Name-based Replication Priorities in Disaster Cases.” In: *2014 Proceedings IEEE INFOCOM Workshops, Toronto, ON, Canada, April 27 - May 2, 2014*. IEEE, 2014, pp. 434–439 (page 103).
- [Res18] Eric Rescorla. “The Transport Layer Security (TLS) Protocol Version 1.3.” In: *RFC* 8446. 2018, pp. 1–160 (page 8).
- [RKS15] Björn Richerzhagen, Boris Koldehofe, and Ralf Steinmetz. “Immense Dynamism.” In: *German Research* 37.2. 2015, pp. 24–27 (page 29).
- [RLM18] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. “Mobile Edge Computing, Fog et al.: A Survey and Analysis of Security Threats and Challenges.” In: *Future Generation Computer Systems* 78.Part 2. 2018, pp. 680–698 (pages 20, 35).
- [Roy+16] Aritra Roy, Supriyo Mahanta, Mallika Tripathy, Sagarika Ghosh, and Sauvik Bal. “Health Condition Identification of Affected People in Post Disaster Area Using DTN.” In: *Proceedings of the IEEE 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON 2016)*. IEEE, 2016, pp. 1–3 (page 103).
- [Sam+11] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. “EnerJ: Approximate Data Types for Safe and General Low-power Computation.” In: *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2011)*. ACM, 2011, pp. 164–174 (page 88).
- [Sap+16] Marco Sapienza, Ermanno Guardo, Marco Cavallo, Giuseppe La Torre, Guerrino Leombruno, and Orazio Tomarchio. “Solving Critical Events through Mobile Edge Computing: An Approach for Smart Cities.” In: *Proceedings of the 2016 IEEE International Conference on Smart Computing (SMARTCOMP 2016)*. IEEE, 2016, pp. 1–5 (page 154).
- [Sat+09] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Cáceres, and Nigel Davies. “The Case for VM-Based Cloudlets in Mobile Computing.” In: *IEEE Pervasive Computing* 8.4. 2009, pp. 14–23 (page 18).
- [Sat17] Mahadev Satyanarayanan. “The Emergence of Edge Computing.” In: *IEEE Computer* 50.1. 2017, pp. 30–39 (page 154).

- [Sch+09] R. Schwarzkopf, M. Schmidt, N. Fallenbeck, and B. Freisleben. “Multi-layered Virtual Machines for Security Updates in Grid Environments.” In: *Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2009)*. IEEE, 2009, pp. 563–570 (page 44).
- [Sch+11] Matthias Schmidt, Lars Baumgärtner, Pablo Graubner, David Böck, and Bernd Freisleben. “Malware Detection and Kernel Rootkit Prevention in Cloud Computing Environments.” In: *Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2011)*. 2011, pp. 603–610.
- [Sch+15a] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann. “OpenSDWN: Programmatic Control over Home and Enterprise WiFi.” In: *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR 2015)*. ACM, 2015, 16:1–16:12 (pages 69, 149).
- [Sch+15b] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, A. Feldmann, and R. Riggio. “Programming the Home and Enterprise WiFi with OpenSDWN.” In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM ’15)*. ACM, 2015, pp. 117–118 (page 149).
- [Sch+17] J. Schulz-Zander, C. Mayer, B. Ciobotaru, R. Lisicki, S. Schmid, and A. 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 149).
- [Sch16] Julius Schulz-Zander. “Realizing Software-defined Wireless Networks.” PhD thesis. Technical University of Berlin, Germany, 2016 (page 10).
- [Sha+06] 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*. Vol. 38. 1. ACM. 2006, pp. 456–457 (page 27).
- [She+15] Haichen Shen, Aruna Balasubramanian, Anthony LaMarca, and David Wetherall. “Enhancing Mobile Apps to Use Sensor Hubs Without Programmer Effort.” In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2015)*. ACM, 2015, pp. 227–238 (page 129).
- [Shi+18] Yoshitaka Shibata, Masaki Otomo, Goshi Sato, and Noriki Uchida. “Efficient Load Balancing and Integration Protocols for Mobile Cloud Computing in Disaster Situations.” In: *Proceedings of the 12th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS-2018)*. Vol. 772. Advances in Intelligent Systems and Computing. Springer, 2018, pp. 228–239 (page 154).
- [Sif+14] Manolis Sifalakis, Basil Kohler, Christopher Scherb, and Christian F. Tschudin. “An Information Centric Network for Computing the Distribution of Computations.” In: *Proceedings of the 1st International Conference on Information-Centric Networking (ICN 2014)*. ACM, 2014, pp. 137–146 (pages 91, 103).
- [Sil17] Mark Silberstein. “OmniX: An Accelerator-centric OS for Omni-programmable Systems.” In: *Proceedings of the 16th Workshop on Hot Topics in Operating Systems, HotOS 2017, Whistler, BC, Canada, May 8-10, 2017*. ACM, 2017, pp. 69–75 (pages 28, 171).

- [SKZ08] S. Srikantaiah, A. Kansal, and F. Zhao. “Energy Aware Consolidation for Cloud Computing.” In: *Proceedings of the 2008 International Conference on Power Aware Computing and Systems (HotPower 2008)*. USENIX Association, 2008, pp. 10–10 (pages 43, 56).
- [Soy+12] Tolga Soyata, Rajani Muraleedharan, Colin Funai, Minseok Kwon, and Wendi B. Heinzelman. “Cloud-vision: Real-time Face Recognition Using a Mobile-Cloudlet-Cloud Acceleration Architecture.” In: *Proceedings of the 2012 IEEE Symposium on Computers and Communications (ISCC 2012)*. IEEE, 2012, pp. 59–66 (page 103).
- [SPR05] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. “Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks.” In: *Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-tolerant Networking (WDTN 2005)*. ACM, 2005, pp. 252–259 (page 103).
- [SSK14] L. Sun, S. Sen, and D. Koutsonikolas. “Bringing Mobility-Awareness to WLANs Using PHY Layer Information.” In: *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT 2014)*. ACM, 2014, pp. 53–66 (page 133).
- [SSS14] Julius Schulz-Zander, Nadi Sarrar, and Stefan Schmid. “AeroFlux: A Near-Sighted Controller Architecture for Software-Defined Wireless Networks.” In: *Open Networking Summit 2014 - Research Track (ONS 2014)*. USENIX Association, 2014 (page 69).
- [ST16] Maarten van Steen and Andrew S. Tanenbaum. “A Brief Introduction to Distributed Systems.” In: *Computing* 98.10. 2016, pp. 967–1009 (page 32).
- [Sta15] William Stallings. “Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud”. Addison-Wesley Professional, 2015 (pages 9, 14).
- [Ste+] Artur Sterz, Pablo Graubner, Matthias Schulz, Robin Klose, Daniel Wegemer, Matthias Hollick, Mira Mezini, and Bernd Freisleben. “ReactiFi: Reactive Programming of Wi-Fi Firmware on Mobile Devices.” In: *Submitted 2018* (pages xiii, xiv, 131, 153).
- [Ste+15] Ralf Steinmetz, Melanie Holloway, Boris Koldehofe, Björn Richerzhagen, and Nils Richerzhagen. “Towards Future Internet Communications - Role of Scalable Adaptive Mechanisms.” In: *Proceedings of the Academia Europaea 27th Annual Conference: Symbiosis - Synergy of Humans & Technology (SIU)*. Academia Europaea, 2015, pp. 59–61 (page 29).
- [Sto+17] D. Stohr, A. Frömmgen, A. Rizk, M. Zink, R. Steinmetz, and W. Effelsberg. “Where are the Sweet Spots? A Systematic Approach to Reproducible DASH Player Comparisons.” In: *ACM Multimedia*. ACM, 2017, pp. 1113–1121 (page 141).
- [Sur+12] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazão. “Towards Programmable Enterprise WLANs with Odin.” In: *Proceedings of the 1st Workshop on Hot Topics in Software-defined Networks (HotSDN@SIGCOMM 2012)*. ACM, 2012, pp. 115–120 (page 69).
- [SWH16] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. “NexMon: A Cookbook for Firmware Modifications on Smartphones to Enable Monitor Mode.” In: *CoRR* abs/1601.07077. 2016. arXiv: 1601.07077 (page 12).

- [Syr+15] Dimitris Syrivelis, George Iosifidis, Dimosthenis Delimpasis, Konstantinos Chou-nos, Thanasis Korakis, and Leandros Tassioulas. “Bits and Coins: Supporting Col-laborative Consumption of Mobile Internet.” In: *Proceedings of the 2015 IEEE Con-ference on Computer Communications (INFOCOM 2015)*. IEEE, 2015, pp. 2146–2154 (page 69).
- [Tar+14] Sasu Tarkoma, Matti Siekkinen, Eemil Lagerspetz, and Yu Xiao. “Smartphone Energy Consumption: Modeling and Optimization”. Cambridge University Press, 2014 (pages 2, 25).
- [Tin+12] Ilenia Tinnirello, Giuseppe Bianchi, Pierluigi Gallo, Domenico Garlisi, Francesco Giuliano, and Francesco Gringoli. “Wireless MAC processors: Programming MAC Protocols on Commodity Hardware.” In: *Proceedings of the 31st IEEE Annual Inter-national Conference on Computer Communications (INFOCOM 2012)*. IEEE, 2012, pp. 1269–1277 (page 149).
- [TNR00] J. Y. F. Tong, David Nagle, and Rob A. Rutenbar. “Reducing Power by Optimizing the Necessary Precision/Range of Floating-point Arithmetic.” In: *IEEE Trans. VLSI Syst.* 8.3. 2000, pp. 273–286 (page 88).
- [Tör+06] B. Ugur Töreyn, Yigithan Dedeoglu, Ugur Güdükbay, and A. Enis Çetin. “Com-puter Vision Based Method for Real-time Fire and Flame Detection.” In: *Pattern Recognition Letters* 27.1. 2006, pp. 49–58 (page 94).
- [TP91] Matthew A. Turk and Alex Pentland. “Face Recognition Using Eigenfaces.” In: *Proceedings of the 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 1991)*. IEEE, 1991, pp. 586–591 (page 83).
- [TS13] Christian F. Tschudin and Manolis Sifalakis. “Named Functions for Media Delivery Orchestration.” In: *Proceedings of the 20th International Packet Video Workshop (PV 2013)*. IEEE, 2013, pp. 1–8 (page 91).
- [TS14] C. Tschudin and M. Sifalakis. “Named Functions and Cached Computations.” In: *Proceedings of the 11th IEEE Consumer Communications and Networking Conference (CCNC 2014)*. 2014, pp. 851–857 (pages 91, 102, 103).
- [TT08] Barry N Taylor and Ambler Thompson. “NIST Special Publication 330: The Inter-national System of Units (SI)”. NIST Special Publication. 2008 (page 21).
- [TUY14] Mohsen Nader Tehrani, Murat Uysal, and Halim Yanikomeroglu. “Device-to-Device Communication in 5G Cellular Networks: Challenges, Solutions, and Future Direc-tions.” In: *IEEE Communications Magazine* 52.5. 2014, pp. 86–92 (page 68).
- [TW17] Thomas N. Theis and H.-S. Philip Wong. “The End of Moore’s Law: A New Begin-ning for Information Technology.” In: *Computing in Science and Engineering* 19.2. 2017, pp. 41–50 (page 1).
- [Urg+15] Rahul Urgaonkar, Shiqiang Wang, Ting He, Murtaza Zafer, Kevin S. Chan, and Kin K. Leung. “Dynamic Service Migration and Workload Scheduling in Edge-Clouds.” In: *Performance Evaluation* 91. 2015, pp. 205–228 (page 39).
- [V+00] Amin Vahdat, David Becker, et al. “Epidemic Routing for Partially Connected Ad hoc Networks”. Technical Report. CS-200006, Duke University, 2000 (page 103).

- [Van+16] Mathy Vanhoef, Célestin Matte, Mathieu Cunche, Leonardo S. Cardoso, and Frank Piessens. “Why MAC Address Randomization is Not Enough: An Analysis of Wi-Fi Network Discovery Mechanisms.” In: *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (AsiaCCS 2016)*. ACM, 2016, pp. 413–424 (page 126).
- [VAN08] Akshat Verma, Puneet Ahuja, and Anindya Neogi. “pMapper : Power and Migration Cost Aware Application Placement in Virtualized Systems.” In: *Proceedings of the ACM/IFIP/USENIX 9th International Middleware Conference (Middleware 2008)*. Vol. 5346. Lecture Notes in Computer Science. Springer, 2008, pp. 243–264 (page 56).
- [VJ01] Paul A. Viola and Michael J. Jones. “Rapid Object Detection Using a Boosted Cascade of Simple Features.” In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001)*. IEEE, 2001, pp. 511–518 (page 83).
- [VJ04] Paul A. Viola and Michael J. Jones. “Robust Real-Time Face Detection.” In: *International Journal of Computer Vision* 57.2. 2004, pp. 137–154 (page 97).
- [Voo+09] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. “Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation.” In: *Proceedings of the 1st International Conference on Cloud Computing (CloudCom 2009)*. Vol. 5931. Lecture Notes in Computer Science. Springer, 2009, pp. 254–265 (page 24).
- [VTM10] Hien Nguyen Van, Frédéric Dang Tran, and Jean-Marc Menaud. “Performance and Power Management for Cloud Infrastructures.” In: *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD 2010)*. IEEE, 2010, pp. 329–336 (page 56).
- [Wu+17] Di Wu, Dmitri I. Arkipov, Minyoung Kim, Carolyn L. Talcott, Amelia C. Regan, Julie A. McCann, and Nalini Venkatasubramanian. “ADDSSEN: Adaptive Data Processing and Dissemination for Drone Swarms in Urban Sensing.” In: *IEEE Trans. Computers* 66.2. 2017, pp. 183–198 (page 103).
- [YSG17] Z. Yang, J. Schafer, and A. Ganz. “Disaster Response: Victims’ Localization using Bluetooth Low Energy Sensors.” In: *Proceedings of the 16th IEEE International Symposium on Technologies for Homeland Security (HST 2017)*. 2017, pp. 1–4 (page 163).
- [Zha+16] Wuyang Zhang, Yi Hu, Yanyong Zhang, and Dipankar Raychaudhuri. “SEGUE: Quality of Service Aware Edge Cloud Service Migration.” In: *Proceedings of the 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2016)*. IEEE, 2016, pp. 344–351 (page 39).
- [Zho+12] Pengfei Zhou, Yuanqing Zheng, Zhenjiang Li, Mo Li, and Guobin Shen. “IODetector: a Generic Service for Indoor Outdoor Detection.” In: *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys 2012)*. ACM, 2012, pp. 113–126 (page 163).
- [Zim80] 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 7, 8, 29).

## Bibliography

---

- [ZZW16] Anatolij Zubow, Sven Zehl, and Adam Wolisz. “BIGAP - Seamless Handover in High Performance Enterprise IEEE 802.11 Networks.” In: *Proceedings of the 2016 IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)*. IEEE, 2016, pp. 445–453 (page 68).

# Curriculum Vitae

Diese Seite enthält persönliche Daten und ist deshalb nicht Bestandteil der Online-Veröffentlichung.