Smart Distributed Sensing in Adaptive Wireless Networks

Dissertation

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

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

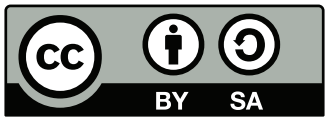
Master of Science (M.Sc.)
Jonas Höchst
geboren in Gießen

Marburg, im Juli 2022

Vom Fachbereich Mathematik und Informatik der Philipps-Universität Marburg (Hochschulkennziffer 1180) als Dissertation am 25. Juli 2022 angenommen.

1. Gutachter: Prof. Dr.-Ing. Bernd Freisleben, Philipps-Universität Marburg
2. Gutachter: Prof. Dr.-Ing. Matthias Hollick, Technische Universität Darmstadt

Tag der Einreichung: 20. Juli 2022
Tag der mündlichen Prüfung: 11. Oktober 2022



Eidesstattliche Erklärung

Ich versichere, dass ich meine Dissertation selbstständig, ohne unerlaubte Hilfe angefertigt und mich dabei keiner anderen als der von mir ausdrücklich bezeichneten Quellen und Hilfen bedient, sowie alle vollständig oder sinngemäß übernommenen Zitate als solche gekennzeichnet habe. Die Dissertation wurde in der jetzigen oder einer ähnlichen Form noch bei keiner anderen in- oder ausländischen Hochschule anlässlich eines Promotionsgesuchs oder zu anderen Prüfungszwecken eingereicht.

Datum

Unterschrift

Abstract

In the recent past, great progress has been made in three technological areas of computer science: sensing, softwarization of networks, and machine learning. Currently, a large variety of sensors is available in many devices, and sensors are getting smaller and more energy-efficient. Software-defined networks are becoming more widespread, achieving low latency and high throughput for emerging applications. Machine learning is very successful in creating and improving services in numerous applications at the edge and in the cloud. There is a great potential in the overlap of these areas: (a) smart processing of sensor data using machine learning methods makes potentially huge amounts of data manageable; (b) adaptive networks support the immediate availability of sensor data in several application areas, and (c) sensor data and machine learning methods are already used in the field of adaptive networks to improve the quality of service.

In this thesis, approaches are presented to improve the quality of service, the quality of experience, and the quality of results of algorithms, protocols, and applications using different sensors and sensor sources. The information analysis cost and the achievable quality of different approaches within the same domain are compared, and a novel classification of smart systems is presented. The main challenge is to balance the information analysis cost generated by additional communication, computation, and storage with the quality improvement achievable by the novel methods. This challenge is addressed by presenting different approaches, algorithms, and systems in the areas of environmental monitoring, adaptive disruption-tolerant networking, and transitional wireless networking.

In the area of smart environmental monitoring, flexible single-board computers are used to realize improvements of various sensing tasks, especially spatial movement and visual / acoustic observation of bats, as well as automated recognition of bird species in audio recordings.

In the area of smart adaptive disruption-tolerant networking, different implementations of disruption-tolerant networks, systems for opportunistic execution of functions and workflows, and novel sensor-based routing algorithms are presented.

Insights from the two areas will be used to develop novel approaches in the area of smart transitional wireless networks for classifying network traffic flow using machine learning, for dynamic announcement intervals in service discovery, and for Wi-Fi connection loss prediction to perform seamless Wi-Fi/cellular handovers.

Deutsche Kurzfassung

In der Vergangenheit sind in drei Technologiebereichen der Informatik große Fortschritte erzielt worden: Sensorik, Softwarisierung von Netzen und maschinelles Lernen. Eine Vielfalt an Sensoren ist mittlerweile in zahlreichen Geräten verfügbar, und Sensoren werden immer kleiner und energieeffizienter. Software-definierte Netze haben sich immer weiter verbreitet, um niedrige Latenzen und hohe Durchsätze für neuartige Anwendungen zu erreichen. Maschinelles Lernen wird sehr erfolgreich in zahlreichen Anwendungen innerhalb der Edge und in der Cloud zur Schaffung und Verbesserung von Diensten eingesetzt. In der Überschneidung dieser Gebiete gibt es große Potenziale: (a) Die intelligente Verarbeitung von Sensordaten mit Hilfe von Methoden des maschinellen Lernens macht riesige Datenmengen überhaupt erst nutzbar. (b) Adaptive Netze ermöglichen die unmittelbare Verfügbarkeit von Sensordaten in vielen Einsatzgebieten. (c) Im Bereich adaptiver Netze werden Sensordaten und Methoden des maschinellen Lernens zur Verbesserung der Dienstgüte bereits eingesetzt.

In dieser Arbeit werden Ansätze präsentiert, um die Dienstgüte, die wahrgenommene Erfahrungsgüte und die Qualität der Ergebnisse von Algorithmen, Protokollen und Anwendungen mit Hilfe unterschiedlicher Sensoren und Sensordatenquellen zu verbessern. Die Kosten der Informationsanalyse und die erreichbare Qualität unterschiedlicher Ansätze innerhalb desselben Bereiches werden verglichen, und eine neuartige Klassifikation smarter Systeme wird präsentiert. Die größte Herausforderung besteht darin, die durch zusätzliche Kommunikation, Berechnungen oder Speicherung erzeugten Kosten der Informationsanalyse abzuwägen hinsichtlich der durch die neuartigen Methoden erreichbaren Qualitätssteigerungen. Diese Herausforderung wird durch unterschiedliche Ansätze, Algorithmen und Systeme in den Bereichen Umwelt-Monitoring, adaptive unterbrechungstolerante Netze und transitionsbasierte drahtlose Netze adressiert.

Im Bereich des smarten Umwelt-Monitorings werden flexible Single-Board Computer eingesetzt, um die Verbesserung verschiedener Sensorik-Aufgaben, insbesondere die räumliche Bewegung und die optische / akustische Beobachtung von Fledermäusen, sowie die automatisierte Erkennung von Vogelarten in akustischen Aufnahmen, zu realisieren.

Im Bereich der smarten adaptiven unterbrechungstoleranten Netze werden unterschiedliche Implementierungen von unterbrechungstoleranten Netzen, Systeme zur opportunistischen Ausführung von Funktionen und Workflows, sowie neuartige sensorbasierte Routing-Algorithmen vorgestellt.

Die gewonnenen Erkenntnisse in den beiden Bereichen werden genutzt, um im Bereich der smarten transitionsbasierten drahtlosen Netze neuartige Ansätze für die Klassifizierung des Netzverkehrsflusses mit Hilfe von maschinellem Lernen, für dynamische Ankündigungsintervalle bei der Ermittlung von Diensten, sowie für die Vorhersage von Wi-Fi-Verbindungsverlusten zur Durchführung nahtloser Übergänge zwischen Wi-Fi und Mobilfunk, zu entwickeln.

Acknowledgments

First of all, I would like to thank my mentor and motivator, Prof. Dr. Bernd Freisleben, for his support during my dissertation. His challenging and motivating nature, as well as his support in various research areas have both motivated me initially to start a doctorate and carried me through many lows. I am grateful for the insights into many different areas and various projects.

I would like to thank Prof. Dr. Matthias Hollick from TU Darmstadt for taking the time to review this work and for the opportunity to work with him and his research group in the SFB MAKI.

I would like to thank all my colleagues of the Distributed Systems Group at the University of Marburg for the fruitful discussions, the productive and often entertaining environment, the open communication, and the great collaboration in paper writing, teaching, project, and administrative work. Over the last seven years, these have been (in alphabetical order): Hicham Bellafkir, Dr. Pablo Graubner, Mechthild Kessler, Nikolaus Korfhage, Matthias Leinweber, Dr. Markus Mühling, Alvar Penning, Falk Schellenberg, Nils Schmidt, Daniel Schneider, Stefan Schulz, Michael Schwarz, Dr. Roland Schwarzkopf, Markus Sommer, Christian Uhl, and Markus Vogelbacher.

During the work on my thesis, I was financially supported by two research projects. I am very grateful for this financial support.

The DFG SFB 1053 MAKI provided an excellent environment for basic research in the area of the future internet. I would especially like to thank the co-authors of my contributions in this area, and the people who made this research environment possible (in alphabetical order): Dr. Michaela Bock, Dr. Alexander Frömmgen, Dr. Patrick Felka, Prof. Dr. Anja Klein, Dr. Katharina Keller, Franz Kuntke, Dr. Manisha Luthra, Julia Müller, Prof. Dr. Mira Mezini, Dr. Tobias Meuser, Prof. Dr. Ralf Steinmetz, and Dr. Denny Stohr.

The LOEWE Nature 4.0 project, funded by the Hessian Ministry of Science and Art (HMWK), introduced me to the topic of nature conservation monitoring and provided exciting tasks, challenges and especially a motivating use case. I would like to thank my co-authors and other team members from Nature 4.0 for their unusual ideas, comprehensive explanations of the tasks in their respective domains and their energy in the many different joint projects (in alphabetical order): Prof. Dr. Nina Farwig, Dr. Nicolas Friess, Dr. Patrick Lieser, Kim Lindner, Prof. Dr. Thomas Nauss, Dr. Christoph Reudenbach, Dr. Sascha Rösner, Dr. Dana G. Schabo, Prof. Dr. Bernhard Seeger, and Julian Zobel. In particular, I would like to thank Jannis Gottwald, who introduced me to the exciting field of bat research.

I would especially like to thank Dr. Lars Baumgärtner. His creativity and improvisational spirit have always been an example to me in my ideas and projects and have ignited my passion for practical research.

I would also like to very much thank Patrick Lampe and Artur Sterz for our time together at the University of Marburg. Since the first semester, we have gone our ways together, experienced many things together, suffered together with our bachelor theses, moved into our first office,

and experienced so many things together away from our work. Without both of you by my side, the professional collaboration and the mutual support, I certainly wouldn't have made it through the time of my doctorate.

Finally, I would like to thank my family, my friends, and my partner. Thank you for making this path possible for me, for supporting me in bad times and for sharing my joy in good times. Thank you Lea, for the open ear, for learning to reflect on myself, for the motivation and strength you give me, and also for the mirror you sometimes held up to me.

My Contributions

I am very grateful to have had the opportunity to work with a variety of people that have fueled my research in different ways and from different perspectives. In the field of distributed systems, computer networks, and ultimately in the development of methods in the ecological field, research is a joint effort of many researchers involved. Publications are created in joint work, implementations are envisioned and developed together, and intellectual capacities are pooled to develop novel concepts and ideas and to discuss results. In addition, students play an important role in implementing ideas or assisting in the evaluation of experiments. Therefore, it is not always possible to attribute the successes to an individual contributor. Since this thesis contains content of original publications, often in verbatim form, it also includes joint and sometimes practically indivisible contributions from colleagues. Therefore, I highlight my specific contributions below.

Chapter 3 presents solely my views and ideas for the research topics addressed in this thesis.

Chapter 4 is based on four joint publications with colleagues of the Nature 4.0 project. The concept of the work presented in Section 4.1 is genuinely my work and was published in publication [Höc+20b]. The design, as well as the implementation of the system was created jointly by Alvar Penning and myself. The evaluation of the system was done by myself. Prof. Dr. Bernd Freisleben and Patrick Lampe reviewed the paper and suggested improvements. The idea behind the open-source software for reliable VHF wildlife tracking [Höc+21] presented in Section 4.2 emerged from discussions of Jannis Gottwald, Patrick Lampe, and myself. Jannis Gottwald contributed the requirements engineering based on his domain knowledge and experience in field work. The design and implementation of the software system was done by myself, the hardware design is based on prior work and was improved by Patrick Lampe and Jannis Gottwald. Julian Zobel implemented the LoRa communications module and provided suggestions for improvements. The evaluation was done jointly by Jannis Gottwald and myself. Both Jannis Gottwald and I contributed equally. Prof. Dr. Thomas Nauss, Prof. Dr. Ralf Steinmetz, and Prof. Dr. Bernd Freisleben reviewed the paper and provided improvements with respect to the writing. The concept, design, and evaluation of BatRack [Got+21] presented in Section 4.3 was done by Jannis Gottwald and Patrick Lampe. Jannis Gottwald proposed, planned, coordinated, and conducted the field campaign and led the writing of the manuscript. The implementation of the system was initially done by Patrick Lampe and later on co-developed and evaluated by Patrick Lampe and myself. Julia Maier, Lea Leister, Tobias Richter, and Betty Neumann deployed BatRack in the field. Prof. Dr. Bernd Freisleben, Dr. Nicolas Friess, and Prof. Dr. Thomas Nauss critically revised the different versions of the system and contributed to its optimization. Bird@Edge [Höc+22b] presented in 4.4 was jointly designed by Hicham Bellafkir, Patrick Lampe, Markus Vogelbacher, Markus Mühling, Daniel Schneider, and myself. The system software and hardware design was done by myself with contributions from Hicham Bellafkir and Patrick Lampe. The neural network architecture design, training, evaluation, and embedded implemnetation were led by Hicham Bellafkir and Markus Vogelbacher with contributions by Markus Mühling and Daniel Schneider. The experimental evaluation of the system was done by myself. Kim Lindner, Dr. Sascha Rösner, Dr. Dana G. Schabo, and Prof.

Dr. Nina Farwig provided domain knowledge and together with Prof. Dr. Bernd Freisleben reviewed the paper and provided valuable feedback for further improving the manuscript.

Chapter 5 is based on six joint publications. The evaluation of the Serval DTN [Bau+16] presented in Section 5.1 system was lead by Dr. Lars Baumgärtner. Patrick Lampe, Nils Schmidt, Stefan Schulz, Artur Sterz, and myself contributed equally to the evaluation of the system, although the contributions were in different areas. My contribution was mainly the analysis of the experimental data. Pablo Graubner contributed the energy-related evaluation. Prof. Dr. Paul Gardner-Stephen, Dr. Jeremy Lakeman, and Prof. Dr. Bernd Freisleben reviewed the paper and provided improvements with respect to the writing. The concept of ONF-DTNs [Gra+18a] presented in Section 5.2 is the work of Dr. Pablo Graubner. I designed and conducted the experiments presented in Section 5.2.6. OPPLOAD [Ste+19] presented in Section 5.3 is the work of Artur Sterz, with Dr. Lars Baumgärtner providing feedback. I contributed the evaluation of the system together with Artur Sterz and helped to write the paper. Section 5.4 is based on publication [Pen+19], which was initially conceptualized by Dr. Lars Baumgärtner and myself. Alvar Penning designed and implemented the system in his bachelor's thesis, which was supervised by myself. Artur Sterz and myself contributed the evaluation of the resulting system and wrote the paper. The publication was intensively discussed with Prof. Dr. Mira Mezini and Prof. Dr. Bernd Freisleben before submission. Section 5.6 is based on publications [Höc+20a] and [Höc+22a]. The concept is genuinely my work, the design and implementations of individual components were contributed by different people. Dr. Lars Baumgärtner contributed the implementation of the rf95modem firmware. Alvar Penning implemented the DTN7 integration. The development of the BlueRa mobile application was done by Artur Sterz. The evaluation of the system was lead by myself, with contributions by Alvar Penning, Franz Kuntke, and Artur Sterz. ProgDTN presented in Section 5.5 is based on publication [Som+22]. The initial concept was conceptualized by me and improved and extended by Markus Sommer and Artur Sterz. The design and implementation was done by Markus Sommer in his master's thesis, which was supervised by myself. Alvar Penning helped with the implementation. The evaluation was done by Artur Sterz and Markus Sommer. Prof. Dr. Bernd Freisleben reviewed, commented, and edited the publication before submission.

Chapter 6 is based on three joint publications with colleagues of the MAKI project. Section 6.1 is based on publication [Höc+17] which was conceptualized by Prof. Dr. Bernd Freisleben and Dr. Lars Baumgärtner. The design, implementation, and evaluation was done by myself. Dr. Lars Baumgärtner provided various feedback and helped writing the manuscript. The publication was intensively discussed with Prof. Dr. Matthias Hollick and Prof. Dr. Bernd Freisleben. The approach presented in Section 6.2 was conceptualized by Dr. Lars Baumgärtner and published in publication [Bau+17]. I contributed parts of the implementation, and designed and supervised the experiments and the evaluation. The energy-related evaluation was contributed by Dr. Pablo Graubner. The paper was intensively discussed with and reviewed by Prof. Dr. Anja Klein and Prof. Dr. Bernd Freisleben. Artur Sterz, Dr. Alexander Frömmgen, Dr. Denny Stohr, and myself jointly developed the concept presented in Section 6.3, which was also published in publication [Höc+19]. The design, implementation, and evaluation of the data collection, data preparation, and implementation of the machine learning model was done by myself, while Artur Sterz aided the design. Artur Sterz contributed the design, implementation, and execution of the evaluation, while I provided helpful feedback. Prof. Dr. Ralf Steinmetz and Prof. Dr. Bernd Freisleben suggested improvements of the paper.

Contents

1 Introduction

In recent years, technical development, miniaturization, and advancements in energy efficiency have lead to the pervasiveness of sensing applications. Smartphones, smart watches, and other wearable devices are already ubiquitous. Other use cases, such as smart homes and sensing in industrial environments, impressively show the emerging deployment of sensing applications. In these use cases, data from heterogeneous sources are used to maximize the quality of sensing applications for users.

A second major achievement is the reconfigurability and softwarization of networks in general and research in the field of adaptive wireless networks in particular. Modern smartphone devices often support multiple radio access technologies (Multi-RAT), which can be used by networked applications for increased service quality in terms of both latency and bandwidth. Adaptations of network properties, however, require information about multiple aspects of protocol internals, network state, and context information, hence there are heterogeneous sensing requirements.

A third driver of modern technology are the improvements in machine learning methods, as well as their deployment in resource-constrained systems. There are many applications in which machine learning is a central component or significantly improves the quality of service. Every major technology company offers digital assistants, e.g., Apple Siri, Microsoft Cortana, Google Assistant, or Amazon Alexa. Car driving assistants and autonomous driving in general are only feasible using a combination of image processing algorithms and prediction mechanisms based on machine learning.

1.1 Motivation

The impact of the ubiquity of sensor data and machine learning methods is already influencing the reconfigurability of adaptive networks, although the potentials have so far not been sufficiently exploited. Classical Internet protocols, such as Ethernet, IP, or TCP, mostly use the sensor data that arises from their domain and limit their control capability in favor of lower complexity. For optimal service quality, however, the sensor sources from other domains or network layers can also be used to further improve service quality. There is already an emerging trend to not only align the configuration of a network with domain-specific information, but also to include other data sources.

Edge computing and fog computing use statistical information about the service usage to bring certain parts of services closer to the users and thus increase service quality. Other areas and

technologies have not yet made use of the broad sensor information available today, or have done so only to a very limited extent. For example, in the area of video streaming, various properties of a video stream, such as the number of buffered blocks, transmission time of a block, and latency are considered, but sensor information of the environment, such as the change of network conditions due to movement of the user, are not yet included in the buffer size or selected video quality. Especially in non-static application areas, such as moving users, temporary infrastructure availability or crisis situations, positive effects can be expected from the aid of additional sensor data.

1.2 Problem Statement

The goal of this thesis is to improve the quality of algorithms, protocols, and applications using different kinds of sensor data and sources. The underlying technical challenges, limitations, and general applicability of this approach will be studied. This goal can be approached by using sensor data from sources outside of their common contexts. Using sensor data effectively and efficiently outside of their common contexts leads to several research questions that need to be addressed.

The first research question is how the great flexibility of single-board computers can be facilitated for smart distributed sensing. Configuration and deployment of such systems either introduces overhead by executing scripts on multiple devices or requires deeper technical understanding when building operating system images for such small computers from scratch. A particular field of interest is environmental sensing, particularly the tracking of small animals, since many of the common challenges occur in this field, i.e., energy constraints and limited communication resources.

The second research question is how to adapt networks, protocols, and applications dynamically with respect to sensed data. Peer-to-peer, delay- and disruption-tolerant networks are applied in the areas of environmental monitoring, and especially in disaster scenarios and emergency response applications. In particular, it must be investigated which network technologies are suitable for which use cases, and which adaptation options can be implemented with these technologies.

The third research question is how sensor data from different sources can be used for adaptations in networks. In particular, the concept of mechanism transitions [Frö+16; Alt+19] allows transitioning between different functionally identical mechanisms used by an application. A transition decision can be made with the help of the many available data sources and thus supports the overall goal of increasing service quality.

1.3 Contributions of this Thesis

In this thesis, the following contributions are presented.

First, advances in smart distributed sensing, in particular in the field of environmental monitoring, are presented:

- PIMOD is presented, a novel software tool and configuration language for configuring operating system images for single-board computer systems. In smart distributed sensing, the configuration and deployment of single-board computer systems can be improved by developing a simple yet comprehensive configuration language and providing a software tool for configuring operating system images.
- For environmental monitoring, the open-source software system tRackIT OS for reliable VHF radio tracking of (small) animals in their wildlife habitat is proposed. It records, stores, analyzes, and transmits detected VHF signals and their descriptive features, e.g., to calculate bearings of signals emitted by VHF radio tags mounted on animals or to perform animal activity classification.
- BatRack is presented, a novel hardware/software system that allows researchers to recognize individual bats and monitor their behavioral patterns to obtain detailed insights into the behavioral ecology of bats.
- Bird@Edge is presented, a novel Edge AI system for recognizing bird species in audio recordings to support real-time biodiversity monitoring. Bird@Edge is based on embedded edge devices operating in a distributed system to enable efficient, continuous evaluation of soundscapes recorded in forests.

Second, various improvements in terms of performance and opportunistic function execution in adaptive disruption-tolerant networks are presented:

- An in-depth experimental evaluation of Serval, an open-source, delay-tolerant, wireless ad-hoc networking systems, is presented. The system can be used to establish a disaster-response communications network spontaneously formed by mobile phones and/or battery powered wireless routers.
- A novel approach to operate information-centric disruption-tolerant networks during emergencies is discussed. Affected people and first responders use their mobile devices to 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.
- OPPLOAD is presented, a novel framework designed for offloading computational workflows in opportunistic networks that provide support for communication in challenging situations. The individual tasks forming a workflow can be assigned to particular remote execution platforms, called workers, either preselected ahead of time or decided just in time where a matching worker will automatically be assigned for the next task in the workflow.
- A novel open source DTN implementation, called DTN7, of the recently released Bundle Protocol Version 7, is presented. DTN7 is written in Go and provides features like memory safety and concurrent execution.
- ProgDTN is presented, a novel approach to support programmable disruption-tolerant networking by allowing network operators to implement and adapt routing algorithms without knowledge of a router's interior workings.

- A novel approach to long-range device-to-device communication via smartphones in crisis scenarios is facilitated through a custom firmware for low-cost LoRa capable microcontroller boards, called rf95modem. Common devices for end users can be enabled to use LoRa through a Bluetooth, Wi-Fi, or serial connection.

Third, insights from sensing and sensor data processing are applied to transitional wireless networks, and consequently improved service quality is achieved.

- A novel approach to unsupervised traffic flow classification using statistical properties of flows and clustering based on a neural autoencoder is presented. In contrast to previous work, the neural autoencoder is used to automatically cluster traffic flows, e.g., into downloads, uploads, or voice calls, independent of the particular network protocols, such as FTP or HTTP(S), used for performing these tasks.
- Several approaches to realize dynamic announcement intervals that facilitate fast reception from at least one other node while keeping the overall communication overhead as low as possible are presented.
- A novel data-driven approach to perform smooth Wi-Fi/cellular handovers on smartphones is presented. The approach relies on data provided by multiple smartphone sensors (e.g., Wi-Fi RSSI, acceleration, compass, step counter, barometric pressure) to predict Wi-Fi connection loss and uses Multipath-TCP to dynamically switch between different connectivity modes.

1.4 Publications

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

1. **Jonas Höchst**, Lars Baumgärtner, Franz Kuntke, Alvar Penning, Artur Sterz, Markus Sommer, and Bernd Freisleben. “Mobile Device-to-Device Communication for Crisis Scenarios Using Low-cost LoRa Modems.” in: *Disaster Management and Information Technology: Professional Response and Recovery Management in the Age of Disasters*. ed. by Hans Jochen Scholl, Eric E. Holdeman, and F. Kees Boersma. Springer Nature, 2022 [Höc+22a]
2. Patrick Lampe, Markus Sommer, Artur Sterz, **Jonas Höchst**, Christian Uhl, and Bernd Freisleben. “Unobtrusive Mechanism Interception: Teaching an Old Dog New Tricks.” in: *2022 IEEE 47th Conference on Local Computer Networks (LCN 2022)*. Edmonton, Canada, Sept. 2022. DOI: 10.1109/LCN53696.2022.9843536 [Lam+22b]
3. Patrick Lampe, Markus Sommer, Artur Sterz, **Jonas Höchst**, Christian Uhl, and Bernd Freisleben. “ForestEdge: Unobtrusive Mechanism Interception in Environmental Monitoring.” in: *2022 IEEE 47th Conference on Local Computer Networks (LCN 2022)*. Edmonton, Canada, Sept. 2022. DOI: 10.1109/LCN53696.2022.9843426 [Lam+22a]
4. **Jonas Höchst**, Hicham Bellafkir, Patrick Lampe, Markus Vogelbacher, Markus Mühling, Daniel Schneider, Kim Lindner, Sascha Rösner, Dana G. Schabo, Nina Farwig, and Bernd Freisleben. “Bird@Edge: Bird Species Recognition at the Edge.” in: *International*

Conference on Networked Systems (NETYS). Springer. May 2022. DOI: 10.1007/978-3-031-17436-0_6 [Höc+22b]

5. Markus Sommer, **Jonas Höchst**, Artur Sterz, Alvar Penning, and Bernd Freisleben. "ProgDTN: Programmable Disruption-tolerant Networking." in: *International Conference on Networked Systems (NETYS)*. Springer. May 2022. DOI: 10.1007/978-3-031-17436-0_13 [Som+22]

6. **Jonas Höchst**, Jannis Gottwald, Patrick Lampe, Julian Zobel, Thomas Nauss, Ralf Steinmetz, and Bernd Freisleben. "tRackIT OS: Open-source Software for Reliable VHF Wildlife Tracking." in: *51. Jahrestagung der Gesellschaft für Informatik INFORMATIK 2021, Berlin, Germany*. LNI. GI, Sept. 2021. DOI: 10.18420/informatik2021-035 [Höc+21]

7. Julian Zobel, Paul Frommelt, Patrick Lieser, **Jonas Höchst**, Patrick Lampe, Bernd Freisleben, and Ralf Steinmetz. "Energy-efficient Mobile Sensor Data Offloading via WiFi using LoRa-based Connectivity Estimations." in: *51. Jahrestagung der Gesellschaft für Informatik, INFORMATIK 2021, Berlin, Germany*. LNI. GI, Sept. 2021. DOI: 10.18420/informatik2021-037 [Zob+21]

8. Jannis Gottwald, Patrick Lampe, **Jonas Höchst**, Nicolas Friess, Julia Maier, Lea Leister, Betty Neumann, Tobias Richter, Bernd Freisleben, and Thomas Nauss. "BatRack: An Open-source Multi-sensor Device for Wildlife Research." in: *Methods in Ecology and Evolution* (July 2021). DOI: 10.1111/2041-210X.13672 [Got+21]

9. Johnny Nguyen, Karl Kesper, Gunter Kräling, Christian Birk, Peter Mross, Nico Hofeditz, **Jonas Höchst**, Patrick Lampe, Alvar Penning, Bastian Leutenecker-Twelsiek, Carsten Schindler, Helwig Buchenauer, David Geisel, Caroline Sommer, Ronald Henning, Pascal Wallot, Thomas Wiesmann, Björn Beutel, Gunter Schneider, Enrique Castro-Camus, and Martin Koch. "Repurposing CPAP Machines as Stripped-down Ventilators." in: *Scientific Reports* 11.1 (June 2021), pp. 1–9. DOI: 10.1038/s41598-021-91673-7 [Ngu+21]

10. Lars Baumgärtner, Alexandra Dmitrienko, Bernd Freisleben, Alexander Gruler, **Jonas Höchst**, Joshua Kühlberg, Mira Mezini, Richard Mitev, Markus Miettinen, Anel Muhamedagic, Thien Duc Nguyen, Alvar Penning, Dermot Pustelnik, Filipp Roos, Ahmad-Reza Sadeghi, Michael Schwarz, and Christian Uhl. "Mind the GAP: Security & Privacy Risks of Contact Tracing Apps." in: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. vol. 1. IEEE. Dec. 2020, pp. 458–467. DOI: 10.1109/TrustCom50675.2020.00069 [Bau+20]

11. **Jonas Höchst**, Alvar Penning, Patrick Lampe, and Bernd Freisleben. "PIMOD: A Tool for Configuring Single-Board Computer Operating System Images." in: *2020 IEEE Global Humanitarian Technology Conference (GHTC 2020)*. Seattle, USA, Oct. 2020, pp. 1–8. DOI: 10.1109/GHTC46280.2020.9342928 [Höc+20b]

12. **Jonas Höchst**, Lars Baumgärtner, Franz Kuntke, Alvar Penning, Artur Sterz, and Bernd Freisleben. "LoRa-based Device-to-Device Smartphone Communication for Crisis Scenarios." in: *17th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2020)*. Blacksburg, Virginia, USA, May 2020 [Höc+20a]

13. Lars Baumgärtner, **Jonas Höchst**, and Tobias Meuser. "B-DTN7: Browser-based Disruption-tolerant Networking via Bundle Protocol 7." in: *2019 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM'19)*. Paris, France, Dec. 2019. DOI: 10.1109/ICT-DM47966.2019.9032944 [BHM19]

14. Alvar Penning, Lars Baumgärtner, **Jonas Höchst**, Artur Sterz, Mira Mezini, and Bernd Freisleben. "DTN7: An Open-Source Disruption-tolerant Networking Implementation of Bundle Protocol 7." in: *18th International Conference on Ad Hoc Networks and Wireless (ADHOC-NOW 2019)*. Esch-sur-Alzette, Luxemburg, Oct. 2019. DOI: 10.1007/978-3-030-31831-4_14 [Pen+19]

15. **Jonas Höchst**, Artur Sterz, Alexander Frömmgen, Denny Stohr, Ralf Steinmetz, and Bernd Freisleben. "Learning Wi-Fi Connection Loss Predictions for Seamless Vertical Handovers Using Multipath TCP." in: *2019 IEEE 44th Conference on Local Computer Networks (LCN 2019)*. **Best Paper Award**. Osnabrück, Germany, Oct. 2019. DOI: 10.1109/LCN44214.2019.8990753. URL: https://umr-ds.github.io/seamcon [Höc+19]

16. Artur Sterz, Lars Baumgärtner, **Jonas Höchst**, Patrick Lampe, and Bernd Freisleben. "OPPLOAD: Offloading Computational Workflows in Opportunistic Networks." in: *2019 IEEE 44th Conference on Local Computer Networks (LCN 2019)*. Osnabrück, Germany, Oct. 2019. DOI: 10.1109/LCN44214.2019.8990775 [Ste+19]

17. Lars Baumgärtner, Patrick Lampe, **Jonas Höchst**, Ragnar Mogk, Artur Sterz, Pascal Weisenburger, Mira Mezini, and Bernd Freisleben. "Smart Street Lights and Mobile Citizen Apps for Resilient Communication in a Digital City." in: *2019 IEEE Global Humanitarian Technology Conference (GHTC 2019)*. Seattle, USA, Oct. 2019. DOI: 10.1109/GHTC46095.2019.9033134 [Bau+19]

18. Manisha Luthra, Boris Koldehofe, **Jonas Höchst**, Patrick Lampe, Ali Haider Rizvi, and Bernd Freisleben. "INetCEP: In-Network Complex Event Processing for Information-Centric Networking." in: *15th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2019)*. Cambridge, UK, Sept. 2019. DOI: 10.1109/ANCS.2019.8901877 [Lut+19]

19. Pablo Graubner, Patrick Lampe, **Jonas Höchst**, Lars Baumgärtner, Mira Mezini, and Bernd Freisleben. "Opportunistic Named Functions in Disruption-tolerant Emergency Networks." in: *ACM International Conference on Computing Frontiers 2018 (ACM CF 2018)*. Ischia, Italy: ACM, May 2018. DOI: 10.1145/3203217.3203234 [Gra+18a]

20. **Jonas Höchst**, Lars Baumgärtner, Matthias Hollick, and Bernd Freisleben. "Unsupervised Traffic Flow Classification Using a Neural Autoencoder." in: *42nd Annual IEEE Conference on Local Computer Networks (LCN 2017)*. Singapore, Oct. 2017. DOI: 10.1109/LCN.2017.57 [Höc+17]

21. Lars Baumgärtner, Pablo Graubner, **Jonas Höchst**, Anja Klein, and Bernd Freisleben. "Speak Less, Hear Enough: On Dynamic Announcement Intervals in Wireless On-demand Networks." in: *13th Conference on Wireless On-demand Network Systems and Services (WONS 2017)*. Jackson Hole, USA, Feb. 2017. DOI: 10.1109/WONS.2017.7888768 [Bau+17]

22. Lars Baumgärtner, Paul Gardner-Stephen, Pablo Graubner, Jeremy Lakeman, **Jonas Höchst**, Patrick Lampe, Nils Schmidt, Stefan Schulz, Artur Sterz, and Bernd Freisleben. "An Experimental Evaluation of Delay-Tolerant Networking with Serval." in: *2016 IEEE Global Humanitarian Technology Conference (GHTC)*. Seattle, USA, Oct. 2016. DOI: 10.1109/GHTC.2016.7857262 [Bau+16]

23. Lars Baumgärtner, **Jonas Höchst**, Matthias Leinweber, and Bernd Freisleben. "How to Misuse SMTP over TLS: A Study of the (In) Security of Email Server Communication." in: *Trustcom/BigDataSE/ISPA, 2015 IEEE*. vol. 1. IEEE. 2015, pp. 287–294. DOI: 10.1109/Trustcom.2015.386 [Bau+15]

1.5 Open Source Software Contributions

During the work on this thesis, the following software was co-developed and released under permissive open source licenses:

1. Bird@Edge OS, an operating system image for Bird Species Recognition at the Edge [Höc+22b]. Available at `https://github.com/umr-ds/BirdEdge`.
2. ProgDTN, a novel approach to support programmable disruption-tolerant networking, based on dtn7-go [Som+22]. Available at `https://github.com/umr-ds/dtn7-go/tree/progdtn`.
3. tRackIT OS, an open-source software for reliable VHF radio tracking of (small) animals in their wildlife habitat [Höc+21]. Available at `https://github.com/Nature40/tRackIT-OS`.
4. BatRack OS, an operating system image for the BatRack multi-sensor device for wildlife research [Got+21]. Available at `https://github.com/Nature40/BatRack`.
5. PIMOD, a tool for reconfiguring Raspberry Pi images with an easy, Docker-like configuration file [Höc+20b]. Available at `https://github.com/Nature40/pimod`.
6. BlueRa, a cross-platform app for connecting to an RF95 modem for chatting over LoRa [Höc+20a; Höc+22a]. Available at `https://github.com/umr-ds/bluera`.
7. rf95modem, a modem firmware for microcontroller boards with a RF95 compatible radio module [Höc+20a; Höc+22a]. Available at `https://github.com/umr-ds/rf95modem`.
8. dtn7-go, a delay-tolerant networking software suite and library based on the Bundle Protocol Version 7 [Pen+19]. Available at `https://github.com/dtn7/dtn7-go`.
9. Seamless Connectivity Demo Application that demonstrates the feasibility of the approach presented in publication [Höc+19]. Available at `https://github.com/umr-ds/seamcon-SeamlessDemo`.

1.6 Organization of this Thesis

This thesis is organized as follows:

Chapter 2 introduces topics fundamental for the research in this thesis.

Chapter 3 gives an overview of the work presented in this thesis. A categorization of smart systems as well as challenging areas covered in the following chapters are explained.

Chapter 4 includes research results obtained to provide efficient and effective implementations in smart environmental monitoring. A software tool for configuring single-board computer systems and an approach for tracking bats are discussed. A novel software-defined radio-based approach for automated signal detection of VHF radio tracking tags, and an open-source multi-sensor device are proposed. Finally, an approach for bird species recognition at the edge is presented.

Chapter 5 presents work to achieve smart adaptive disruption-tolerant networking. The evaluation of delay- and disruption-tolerant network approaches are discussed, and various improvements, such as opportunistic offloading and workflow scheduling in such networks, are presented. An approach to support programmable disruption-tolerant networking is proposed. The versatility and adaptivity of long range wireless communication is explored, and hybrid DTN/LoRa approaches are presented.

Chapter 6 presents approaches to improve service quality in smart transitional wireless networking. Unsupervised traffic flow classification as a decision basis for transitions, implementations of dynamic announcement interval algorithms, and a machine-learning approach to Wi-Fi connection loss predictions are discussed.

Chapter 7 concludes the thesis and discusses possible areas of future work.

2
Fundamentals

In this chapter, fundamental concepts and technologies that are used throughout this thesis are discussed. First, the term *smart systems* is introduced. Then, the term *adaptive wireless networks* and the related subcategory of *transitions* are discussed. Finally, different quality metrics based on results, service quality, and user-perceived quality are described.

2.1 Smart Systems

In his visionary article, Mark Weiser describes 'The Computer for the 21st Century' [Wei91] as the foundation for today's research field of ubiquitous computing.

> *'The real power of the concept comes not from any one of these devices; it emerges from the interaction of all of them. The hundreds of processors and displays are not a 'user interface' like a mouse and windows, just a pleasant and effective 'place' to get things done [Wei91].'*

Ubiquitous computing touches various fields of computer science and information and communication technology (ICT), including distributed systems, (wireless) sensor networks, context-aware systems, and artificial intelligence.

In the context of ubiquitous computing, Poslad defines a smart entity of a system as 'active, digital, networked, operating to some extent autonomously, is reconfigurable and has local control of the resources it needs such as energy, data storage, etc.' [Pos11]. In Figure 2.1, three device trends are identified and that lead to smarter devices, environments, and interactions, and ultimately led to the rise of ubiquitous computing. With the dawn of the smartphone era, smart devices entering people's homes, and smart cities solving infrastructure tasks with the help of smart digital devices, we are seeing these predictions to come true.

In more recent years, the term smartness has been used in various contexts and is particularly familiar in smart homes and smart cities, where it is used to describe the use of smart devices to make the lives of people more convenient. Furthermore, the terms smart industry and smart service systems have entered the literature to consider the use of cyber-physical systems and even people within such systems. From a systematic literature review, Romero et al. derive the specifics of a smart system [Rom+20]:

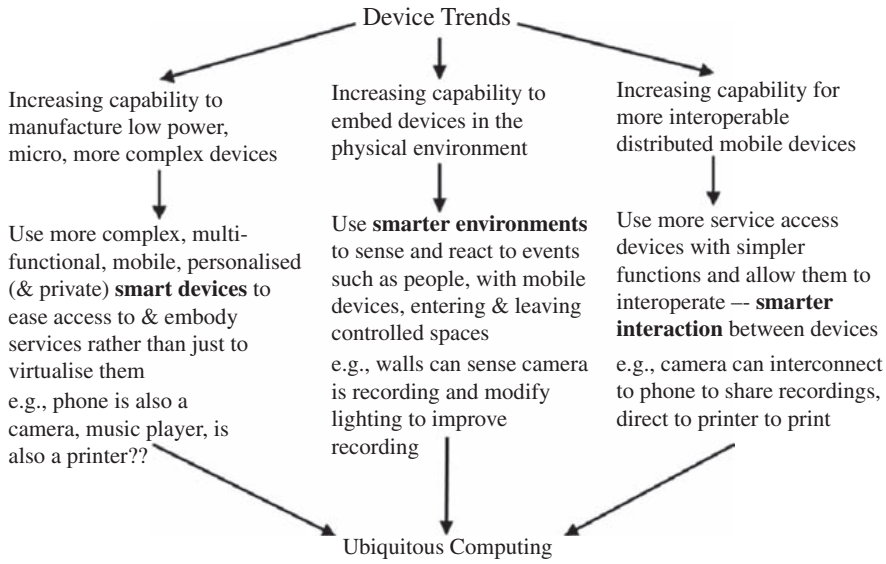


Figure 2.1: Three different models of ubiquitous computing: smart terminal, smart interaction, and smart infrastructure, as defined by Poslad [Pos11].

> *'Smart systems […] are able to self-organise and be aware of the context, to provide communication between their elements. Moreover, they are able to learn, reason, perceive themselves and their surroundings, control their environment, and have embedded knowledge to be used for making decisions [Rom+20].'*

Medina-Borja [Med15] defines smart service systems in the editorial column of the Journal of Service Science using a definition of the NSF [Fou14]:

> *'A 'smart' service system is a system capable of learning, dynamic adaptation, and decision making based upon data received, transmitted, and/or processed to improve its response to a future situation. The system does so through self-detection, self-diagnosing, self-correcting, self-monitoring, self-organizing, self-replicating, or self-controlled functions. These capabilities are the result of the incorporation of technologies for sensing, actuation, coordination, communication, control, etc. The system may exhibit a sequence of features such as detection, classification, and localization that lead to an outcome occurring within a reasonable time [Fou14].'*

Based on this definition, further definitions in the literature, and an analysis of a large selection of examples, Alter derives four categories of smartness, which are further broken down into 23 dimensions [Alt20]. Within the dimensions, a continuous scale is applied, which is described by five levels of smartness: not smart at all, scripted execution, formulaic adaptation, creative adaptation, and unscripted or partially scripted invention. Figure 2.2 shows the categories and dimensions according to the definition of smartness in Alter's paper.

In the context of this thesis, we will use the definition of smartness based on Medina-Borja's definition. Hence, a smart system in the context of this thesis is one that is capable of learning, dynamic adaptation, and decision making based upon data. In addition to this, Alter's scale for smartness is used as an inspiration for the classification system presented in Chapter 3.

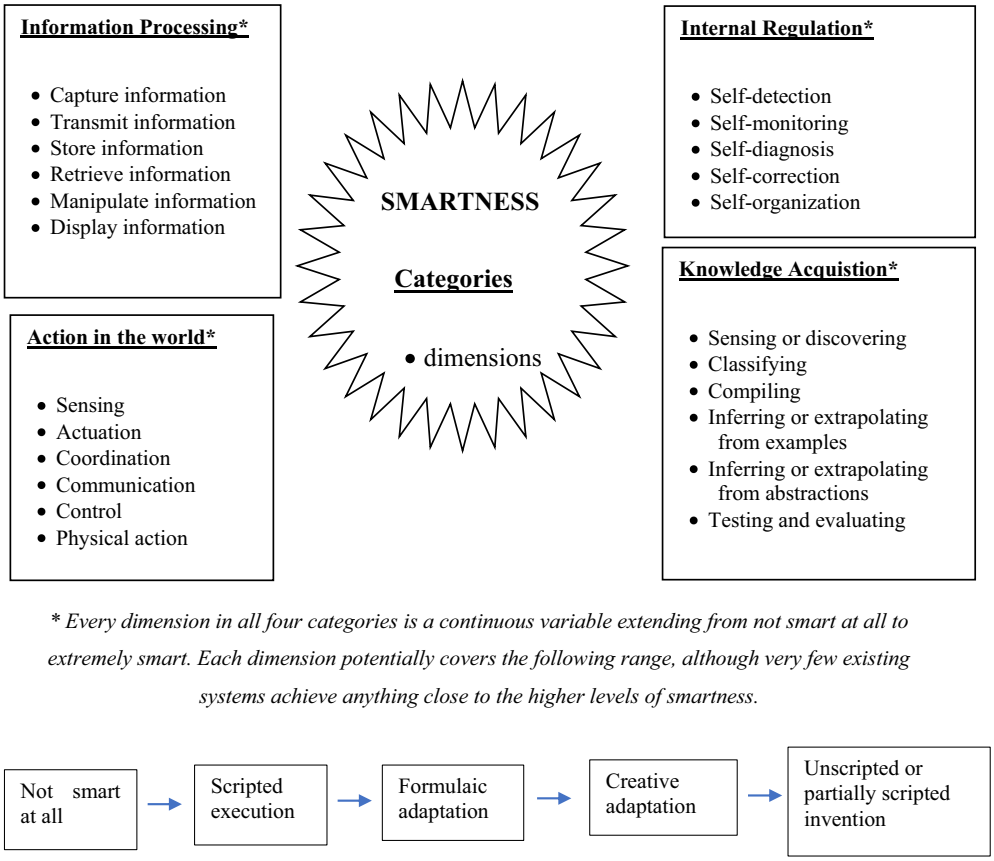


Figure 2.2: Dimensions of smartness for systems, services, and devices, as defined by Alter [Alt20]

2.2 Smart Distributed Sensing

According to the common definition by Tanenbaum [VT02], a distributed system is a system whose components are distributed on different networked devices that communicate with each other and coordinate their actions by message passing. The term smart distributed sensing is defined in this thesis as follows.

> Smart distributed sensing is the combination of a number of autonomously operating devices and sensors that perform a sensing task in a coordinated manner. The systems covered by the term can have a tendency towards a smart system, i.e., be autonomous and only contribute to a distributed system through individual aspects, or be comparatively non-smart, however contribute to the sensing task through the distributed aspects. Particularly noteworthy is the interplay between sensing as the task of the systems under consideration and the concept of smartness, in which sensors and information from the local device itself enable adaptation.

Smart distributed sensing is one of the building blocks of the Nature 4.0 research project, which is funded by the Hessian Ministry of Science and Art (HMWK):

> *'The project combines expert surveys by nature conservationists, remote sensing, and a network of environmental sensors, which are integrated into stationary units as well as attached to unmanned aerial vehicles, rovers, or animals. By utilizing powerful data integration and analysis methods, Nature 4.0 will enable researchers to effectively observe landscapes through a set of diverse lenses. [...] Nature 4.0 will establish new methods and protocols in the field of comprehensive environmental monitoring by combining traditional sampling, remote sensing, and automated measurement stations [Fri+19].'*

2.3 Adaptive Wireless Networks

In recent years, Software-defined (Wireless) Networking (SD(W)N) has gained considerable attention, especially because rapid reconfiguration of networks allows the use of novel protocols [Xia+14]. In addition to common implementations of SDN concepts, i.e. OpenFlow, more unconventional concepts such as delay- and disruption-tolerant networking, Long Range (LoRa) and decentralized networks are also experiencing softwarization and thus increased adaptivity.

The concept of mechanism transitions in communications systems was developed by the Collaborative Research Center Multi-Mechanism Adaptation for the Future Internet (MAKI) funded by the German National Science Foundation [Frö+16; Alt+19]. While many protocols used in networks and on the Internet already function adaptively and adapt to their environment at runtime, the adaptation of the protocol or mechanism itself to another more suitable mechanism is usually not possible. The goal of MAKI is to explore techniques for transitioning between mechanisms at runtime to provide additional flexibility and adaptivity, especially in challenging situations. Frömmgen et al. defines a transition inside a communications system as follows:

> *'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, without causing an error condition in any dependent mechanism [Frö+16].'*

An illustrative example is a transition between the well-known transport mechanism TCP and the standard QUIC or HTTP/3, which has become widespread in recent years. If an adaptive wireless network or system triggers a change of the transport mechanism from TCP to QUIC or vice versa based on a change of external conditions or internal states, this is a transition. However, mechanism transitions are not limited to the transport layer, but can be applied to all seven layers of the ISO OSI model, as well as in other areas of information systems, such as the calculation or storage of data, both in the selection of the algorithm used and the location of the calculation or storage [Gra19].

In the context of this thesis, the term adaptive wireless networks describes networks that adapt by means of conventional adaptation within specific mechanisms or protocols or by means of

mechanism transitions. Examples of such networks can be found in the areas of delay- and disruption-tolerant networks, more specifically in the adaptive allocation of computations of individual tasks of a workflow (Section 5.3), or in the opportunistic computation of partial results in information-centric networks (Section 5.2).

2.4 Quality of Service / Experience / Result

Different approaches can be used to evaluate algorithms and systems, which can basically be divided into three categories. Figure 2.3 provides an overview on the terms.

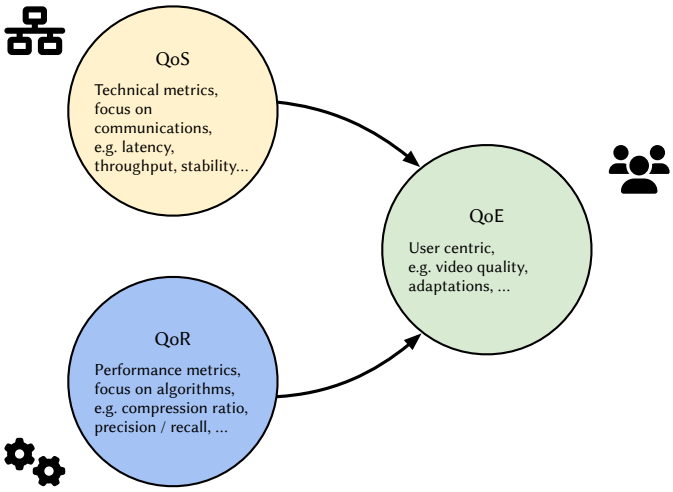


Figure 2.3: Quality of Result / Service / Experience

The term Quality of Result (QoR) is traditionally used in the field of technological processes, and will be used in the context of this thesis as a collective term to evaluate the result itself. For example, QoR refers to the sheer number of available measurement data or their resolutions, e.g. the accuracy and frequency of a GPS position, or the frequency of a temperature measurement. Especially in the example of data-driven approaches, such as machine learning, the usual evaluation criteria, such as precision and recall, as well as f-scores belong to the category of Quality of Result. In contrast to QoS and QoE, QoR does not evaluate the technical or perceived availability of a system, but the quality of the system itself.

The development and testing of systems, especially in the academic field, focuses on technical metrics. The term Quality of Service (QoS) originates from an ITU definition in the field of telephony, but has since been used in different areas to describe different technical metrics. Depending on the domain and the specific requirements, the term can refer to a variety of evaluation functions, all of which are characterized by their technical measurability. These include the measured throughput or the delay of a connection, the duration of a calculation or a compression rate when storing data.

The term Quality of Experience (QoE), on the other hand, follows a holistic concept and takes the user's perception as the basis for evaluating a system [FHT10]. This broader view creates a link between user perception and technical background and enables the optimization of a

technical system in terms of perceived quality. Especially in the area of video streaming, the evaluation of quality based on the user experience has become common. Examples of QoE metrics can be the visual quality of a video, based on resolution and bit rate, but also the number of stalling / buffering events and the frequency of quality adjustment.

In principle, the three terms can be used differently in different domains and can also merge into one another depending on the domain, for example when the quality of experience is approximated using technical QoS metrics. In particular, when smart distributed sensing is used in adaptive wireless systems, QoR and QoS also serve to distinguish the quality of the communication system and the sensing system.

3

Categorizing Smart Systems

In this chapter, a categorization of smart systems overarching the three areas of environmental monitoring, adaptive disruption-tolerant networking, and transitional wireless networking is discussed. First, the basis for determining quality and information analysis cost of systems in a domain is provided. Then, for each of the three areas, the contributions of this thesis are classified according to the proposed schema.

3.1 Determining Quality and Information Analysis Cost

In principle, the quality of a system or algorithm can be quantified in different ways. Quality of Service (QoS) describes technical metrics for measuring quality, such as the delay or throughput in a communication link. Quality of Experience (QoE) describes the experience of a user, e.g., the user's satisfaction with the system. Quality of Result (QoR) describes the quality of the results of an algorithm, i.e., the objective of the algorithm itself, e.g., the accuracy of a position finding or the temporal resolution of a measurement series.

On the other hand, there are information analysis costs, i.e., costs occurring in a system or a solution in order to achieve an increase of quality. These costs can also be represented in a variety of ways. The most common form are technical metrics, such as the number of CPU cycles required to calculate a result, or more practically, the computing time required. In addition to the pure computing time, costs can also be incurred in the area of data storage, for example, if an entire measurement series must be kept available. Finally, in distributed systems, the costs of communication must also be considered, e.g., when information from many participants must first be collected for decision-making.

The consideration of both quality and information analysis cost is usually based on the problem or problem domain. The comparison of systems on the basis of this consideration is therefore only meaningful within the same problem or the same domain. Nevertheless, common characteristics can be identified across different domains.

Basically, it can be stated that there is a relationship between the use of resources and the increase in quality of a solution. One challenge is to use additional resources efficiently, i.e., to achieve a disproportionate improvement of the solution with as little effort as possible.

Figure 3.1 shows an overview of categories for evaluating smart systems based on quality achieved and resources used. The x-axis shows the information analysis cost, a metric that can be quantified differently depending on the area of the task. In this figure, it shows the abstract

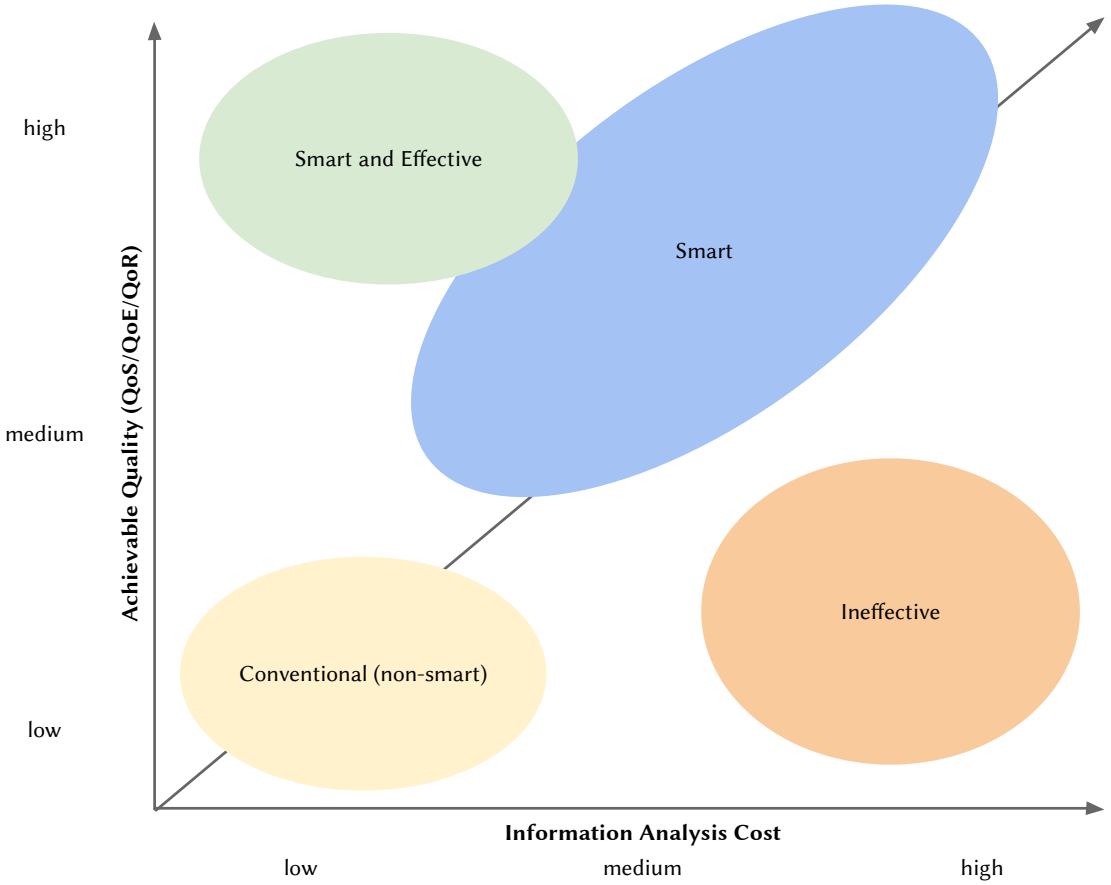


Figure 3.1: Categories of conventional and smart systems emerging from information analysis cost and achievable quality

costs or effort in the areas of communication, computation, and storage. The y-axis represents the achievable quality of a solution, i.e., technical (QoS), user-centric (QoE), or result-centric (QoR) quality metric.

Traditionally, simple algorithms with comparatively low cost have been used in many areas, i.e., conventional, non-smart solutions indicated by the yellow area. In particular, the limitations of hardware and, to some extent, software have led to the emergence of simple solutions with acceptable quality.

Due to the availability of resources, more and more smart solutions have emerged in recent years that use available resources to increase the quality of a solution. Newly available approaches to algorithms also allow this improvement, such as the use of machine learning and artificial intelligence. In Figure 3.1, these smart solutions are colored in blue.

A third category is that of ineffective solutions, colored in orange, where additional information analysis costs are introduced that do little or nothing to improve quality. Some ineffective

solutions are interesting from an academic perspective, because they contain new approaches to solving problems that may be further improved. In principle, however, ineffective solutions can also be a consequence of wrong decisions in technology selection or insufficient development effort, for example, when a solution does not exploit the available potential, such as in the case of insufficiently trained machine learning approaches.

The last and most valuable category is that of smart and effective solutions, colored in green. This category includes solutions that experience a high increase in quality due to a slight increase of information analysis cost. It usually takes novel approaches and ideas to arrive at smart and effective solutions. The use of new technologies, additional data, or the use of other resources may be useful to achieve a high increase of quality.

The quantification of both the information analysis cost and the achievable quality is crucial for the identification of practical solutions. The baseline for comparing different solutions in the approach presented here is not just a single metric, but the quotient of cost and quality. Both cost and quality can be quantified based on different metrics or combinations of different metrics, but must be quantified in the same way for comparing solutions.

3.2 Environmental Monitoring

In the area of environmental monitoring, smart distributed sensing can particularly be used to improve the QoR of systems. Conventional, non-smart sensing systems in the field of environmental monitoring require either the manual acquisition of the measurements themselves, or at least the manual collection of the measured data. More recent developments include the use of smart systems that perform demand-oriented measurements or use measurements from multiple stations to derive high-quality measurements.

Figure 3.2 compares achievable quality and information analysis cost of the presented systems in Chapter 4.

First, tRackIT OS is a smart distributed sensing system that allows monitoring and observation of bats based on Very High Frequency (VHF) tags attached to individual bats (Section 4.2). Compared to manual VHF telemetry, which is the de-facto standard for VHF telemetry, tRackIT OS uses a combination of wireless sensor nodes and a wireless network to perform the analysis. The information analysis cost of tRackIT OS is higher compared to manual telemetry in terms of computation, since the detection of signals requires local computation and communication resources, which are accomplished by people in the manual case. The QoR is orders of magnitudes better compared to the manual telemetry, since observations are accomplished in seconds instead of a few observations per night in the manual case. Compared to a similar acquisition system of the radio-tracking.eu project, called paur, tRackIT OS uses communication and computing resources to take advantage of an intelligent distributed acquisition system. While paur collects signals locally to the best of its ability, with signals missing or delayed for up to minutes, tRackIT OS constantly ensures the correct functionality of all components, supports detection without delays, and transmits detected signals live to a server system that performs further calculations on the data, i.e., activity classification, position trilateration, and body temperature computations. The additional computational cost required to constantly monitor the individual sensors, as well as the additional effort required to communicate with the server

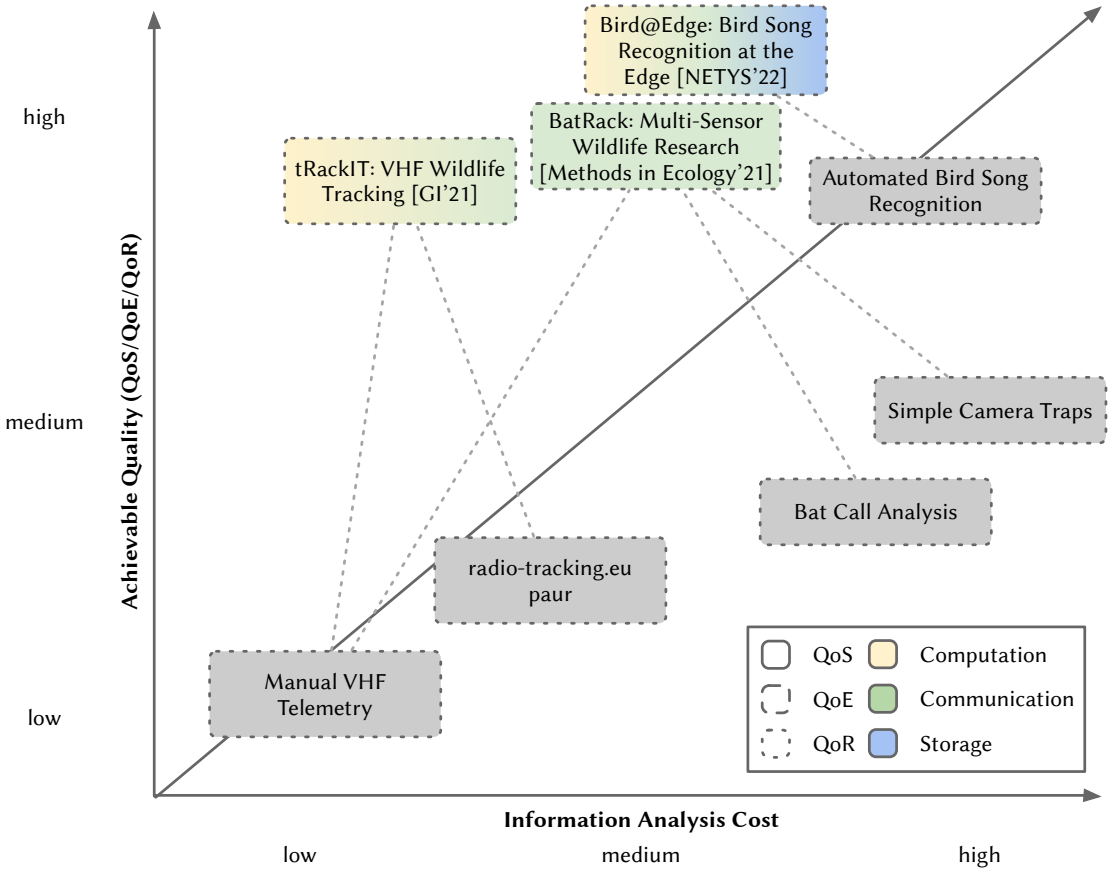


Figure 3.2: Information analysis cost and achievable quality of smart environmental monitoring systems presented in this thesis

system, increase the QoR on the one hand, but also the QoE, since researchers have direct insights into the data and do not have to work with possibly erroneous data months later.

Second, there is BatRack, a hardware/software system that uses VHF signals, ultrasonic audio recordings, and video camera recordings to trigger audio and video recordings of bats to support the direct observation of marked individuals (Section 4.3). BatRack combines three functions that were previously performed independently of each other. The observations with the help of the VHF technology are realized as in tRackIT OS and offer similar advantages as already described above. Acoustic observations of bats have so far been carried out either manually, i.e., by people at a remote site, or with the aid of recording devices that record at fixed time intervals or are triggered by sounds at specific frequencies. The latter method is used for the audio recordings in BatRack. Until now, optical observations were only possible by human observers or by simple photo traps, which, however, have the disadvantage of comparatively poor precision, because they are based on simple motion detectors. Continuous video recordings can be realized for a few locations, but they require immense storage capacities and are therefore only scalable to a limited extent. Through the combination of all three different observation methods, BatRack can be used to monitor and observe bats directly without the need for a human observer and without large overheads in terms of storage and computational power,

e.g., by storing or processing video files without observations. BatRack continuously evaluates the two data sources VHF signals and ultrasonic audio. As soon as bats are detected in one way or another, audio and video recordings are started, and the data is recorded for further analysis by researchers. The combination of these three observation methods leads to much better QoR compared to the non-smart approaches, i.e., more actual observations of bats and less data without meaningful content.

Finally, Bird@Edge is presented, where Edge Computing and Artificial Intelligence (AI) is combined for recognizing bird species in audio recordings to support real-time biodiversity monitoring (Section 4.4). The task of recognizing bird species is conventionally performed by researchers in the field, but there are also systems that have automated the recognition of bird species. Bird@Edge is a smart distributed sensing system that consists of three different components. Multiple Bird@Edge mics record audio data and transmit it to a Bird@Edge station where the actual audio analysis is performed. The results are then transmitted to a server system for further analysis by researchers. A specific improvement of Bird@Edge compared to other automatic bird species recognition systems is the reduction of acquisition costs, since only one Bird@Edge station is needed for up to 10 survey sites (i.e., Bird@Edge mics). The Bird@Edge information analysis cost in terms of storage is reduced, since recorded files do not need to be saved, which is required in conventional field recording approaches. Computational cost is neither reduced nor increased, since the analysis is shifted from the server side to the edge. Cost in terms of communication is also reduced, since the audio files themselves are not transferred over larger distances, but only locally. Besides these cost reductions, the QoR is improved, since larger quantities of data and observation locations become feasible. Also, QoS and likewise QoE are improved, since the data is processed in real-time, and no manual data collection step is required.

3.3 Adaptive Disruption-tolerant Networking

One aspect of smart distributed sensing is in-network processing, where data is processed by a network consisting of smart sensors. In-network processing first requires a network layer, which is suitable for the processing of data. In this thesis, adaptive networks and especially delay- and disruption-tolerant networks (DTNs) are used as network layers, namely the two approaches Serval and DTN7, as presented in Chapter 5. Built on top of DTNs, two approaches of in-network processing are designed, implemented, and evaluated.

First, opportunistic named functions in disruption-tolerant networks are shown, which rely on user-defined interests and on locally optimal decisions based on battery lifetimes and device capabilities, as discussed in Section 5.2. The system allows data sinks to specify interests for data or executed functions on this data. Data sources provide data to the network. The network has the task of executing the function calls partially or completely on the provided data and delivering them to the data sinks. Conventionally, the data would be processed at either the source or the sink, and the DTN would only handle the transmission. However, by performing the functions within the network, there are some advantages, such as the ability to transmit intermediates up to certain points in the network when multiple functions are concatenated and the interests of different data sinks partially match. In addition, participants can decide opportunistically and based on their own capacities which functions they perform and when

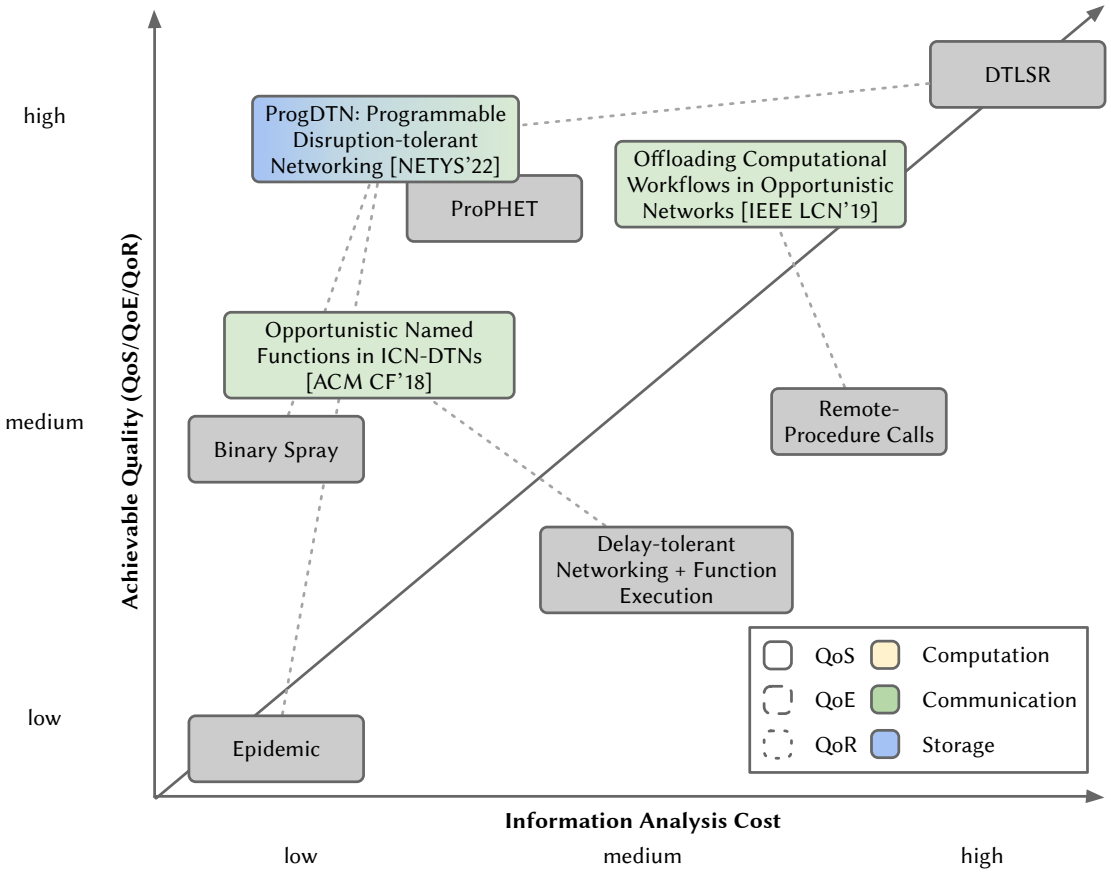


Figure 3.3: Information analysis cost and achievable quality of smart adaptive disruption-tolerant networking approaches presented in this thesis

they only provide forwarding. The function execution usually reduces the amount of data to be transmitted over the network. Even if it is domain-specific, the interests of a data sink are usually very specific, such as the recognized faces on an image, or as in the example of environmental monitoring, a recognized bird species. This reduction in the amount of data transmitted reduces the load on the system and ultimately improves the quality of service, since the transmission of less data is faster. The QoS is increased by the system, since only data of interest is forwarded. Compared to a simple approach based on DTN and regular function execution, the cost in terms of communications is lower, due to the function execution happening earlier in the network. On the other hand, the computational effort can also be reduced, since interests that are requested twice do not have to be completed multiple times on the target computers, but only once in the network.

Second, offloading of computational workflows formed by individual tasks to the opportunistic network is discussed in Section 5.3. The system works in such a way that a client makes a request to the network for the calculation of a workflow, which consists of several individual tasks. It can be selected whether the individual tasks are to be distributed before the calculation or dynamically to individual workers. The system implements the communication and calculations and, in particular, takes care of error handling if one or more nodes cannot perform the

calculations, for example, because they are no longer part of the network. Conventionally, such a task could be implemented with remote procedure calls. However, this requires a permanent connection between the client and the worker, which cannot be assumed, especially in opportunistic networks. A lower information analysis cost is achieved by handling workflow scheduling and status management in the network instead of in the client. Higher quality is mainly achieved by running individual tasks of a workflow in stages and implementing error handling options between those tasks. Workflows that would fail, due to network losses or worker clients leaving the network, are handled by the network and allow successful termination even in disrupted networks.

Third, ProgDTN is presented, a novel approach to support programmable disruption-tolerant networking by allowing network operators to implement and adapt routing algorithms (Section 5.5). Such a routing algorithm can use arbitrary information shared by adjacent nodes to make routing decisions, hence higher cost in terms of communication and storage is required compared to routing algorithms not using contextual information. However, higher QoS is achieved, since the routing algorithm can be adjusted to match the scenario the network is used in, e.g., an emergency response scenario or a smart distributed sensing scenario.

Compared to classical DTN routing approaches, a much better QoS is achieved, since unnecessary transmissions are prevented and congestion and transmission delays are reduced. The primary cost metric is the bundle overhead, i.e., the percentage of metadata bundles. Due to the nature of the approach, the actual improvements depend on the implementation of the network operator. However, for the scenario presented in the section, several improvements can be identified: Regarding information analysis cost, since Epidemic Routing and Binary Spray do not require any additional communication, there is an overhead of 0. As shown in Figure 3.3, while the overall QoS of DTLSR is higher than the other approaches, the costs are disproportionately high. ProgDTN and PRoPHET are comparable in both QoS and cost, as we can draw from the experimental evaluation that ProgDTN reaches an almost 100 % delivery rate compared to about 70 % for PRoPHET while the median delivery time only differs by a few nanoseconds. Depending on the task the QoS weighting may be different, e.g., for a smart distributed sensing scenario only delivery might be important, while for an emergency response scenario both metrics are equally important. In essence, the approach of ProgDTN allows network operators to trade-off information analysis cost for QoS depending on their scenario.

3.4 Transitional Wireless Networking

The third area in which smart distributed sensing is beneficial is the field of transitional wireless networking, where transitions between different network mechanisms are used to improve network performance. The concept of smart distributed sensing is used for transition decisions in many transitional wireless networking systems and algorithms, since multiple sensor sources of different abstraction levels often are superior compared to domain-specific single sources. Figure 3.4 shows the information analysis cost and achievable quality of the presented three different transitional wireless networking systems of Chapter 6.

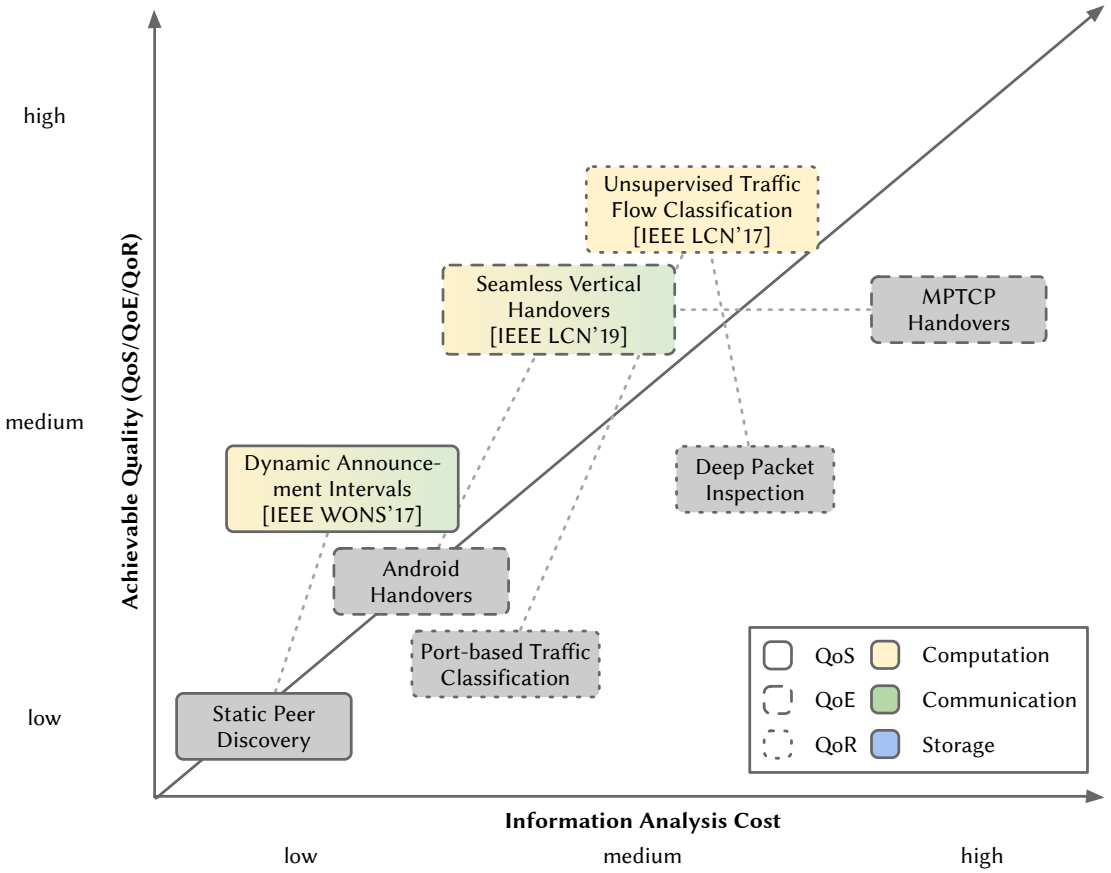


Figure 3.4: Information analysis cost and achievable quality of smart transitional wireless networking approaches presented in this thesis

First, a novel approach to unsupervised traffic flow classification using statistical properties of flows and clustering based on a neural autoencoder is presented in Section 6.1. Conventional alternatives to this approach are port-based classification, which does not allow high quality, because nowadays different protocols and network tasks are performed over the same port and protocols, i.e., HTTP(s) for web traffic, video, interactive content, etc. Another alternative is deep packet inspection (DPI). It aims to identify protocols on the basis of packet headers, which does not allow sufficient results, e.g., when traffic is encrypted. Compared to port-based network traffic classification, the information analysis cost of this approach is higher, since the statistical properties first need to be computed on which the neural network can classify the traffic. Compared to DPI, the information analysis cost is comparable, however DPI's cost is largely dependent on the number of protocols that are inspected. The achievable QoR is higher, since the network can classify flows based on metadata, which are more independent of single values, an implementation or even concrete protocols in some cases.

The second presented approach is using dynamic announcement intervals in the application of peer discovery. In traditional approaches, a static peer discovery interval is used, which is replaced by a dynamic interval based on the number of peers discovered in the approach presented in Section 6.2. Especially in networks where individual participants want to quickly

discover a group of participants, for example, when passing by, it is helpful if peer discovery is designed dynamically. A peer listens to its surroundings and determines how often it announces itself based on the perceived neighboring participants. The information analysis cost in the presented approach is higher, since the client needs to be able to determine the number of peers in proximity. However, the achieved QoS is higher, because while the static announcement interval may work for one general case, a dynamic interval is more suited to match multiple scenarios, e.g., a network with only short encounters of peers or a network where single nodes meet large groups of people.

The last presented approach is realizing seamless vertical handovers by learning Wi-Fi connection loss predictions, which is presented in Section 6.3. In this approach, the data of various sensors of a smartphone are used to learn the connection loss between a smartphone and a Wi-Fi network. The trained model is then applied to trigger proactive transition to cellular networks, if a connection loss is predicted. Regular vertical handover mechanisms rely on signal strength measurements. Depending on the implementation, they reactively switch to cellular networks after the connection loss has happened. The information analysis cost of the presented approach is higher in terms of computation, since multiple sensors values of various parts of the smartphone are read and a trained model is executed to predict the connection loss. The communication costs of the presented approach are lower, since the cellular module can be switched off when it is not needed, and redundant transmissions made by the MPTCP underneath can be avoided. Since the mobile data volume is limited for many users by their contracts, this savings potential is particularly relevant. The QoE is higher, i.e., video stalling and buffering events, as well as frequent quality adaptations could be reduced. Finally, the implementation of Wi-Fi loss prediction in this work is not optimized for energy consumption, although enormous savings can be expected.

4

Smart Environmental Monitoring

In this chapter, novel smart distributed sensing approaches in the field of environmental monitoring are presented.

In smart distributed sensing, the configuration of sensor nodes and networks often requires deeper technical understanding when building such systems using general-purpose single-board computers, such as a Raspberry Pi. In Section 4.1, PIMOD is presented, a software tool for configuring single-board computer systems to allow fast and easy configuration of such computer systems.

In environmental monitoring, VHF wildlife tracking is a common method of observing small animals, including birds and bats. In Section 4.2, a novel sensor system is presented. It replaces manual radio telemetry by distributed sensor nodes built from low-cost commodity-off-the-shelf hardware to allow fine-grained localization of small animals.

The approach is further extended by the work presented in Section 4.3, in which VHF signals, ultrasonic audio recordings, and video camera recordings are analyzed to trigger audio and video recordings of bats to allow direct observation of marked individuals.

Another challenge is real-time biodiversity monitoring, which can be achieved by recognizing bird species in audio recordings. A smart distributed sensing approach to bird species recognition in soundscapes in presented in Section 4.4. The Bird@Edge system is based on embedded edge devices operating in a distributed system to enable efficient, continuous evaluation of soundscapes recorded in forests.

PIMOD[1], *tRackIT OS*[2], *BatRack*[3], and *Bird@Edge*[4] are available via GitHub under open-source licenses.

Figure 4.1 shows the relation of achievable quality and information analysis cost of the contributions presented in this chapter. PIMOD is not shown in the figure, since it is used as a prerequisite and valuable supporting tool to create the software of the three other approaches. Since in the field of environmental monitoring the contributions are methodological contributions, the QoR is the subject of improvement. Since these new methods are intended to replace established, manual methods, especially through smart distributed sensing techniques, improvements in the area of QoS are also achieved, e.g., through the direct availability of data.

[1] `https://github.com/Nature40/pimod`
[2] `https://github.com/Nature40/tRackIT-OS`
[3] `https://github.com/Nature40/BatRackOS`
[4] `https://github.com/umr-ds/BirdEdge`

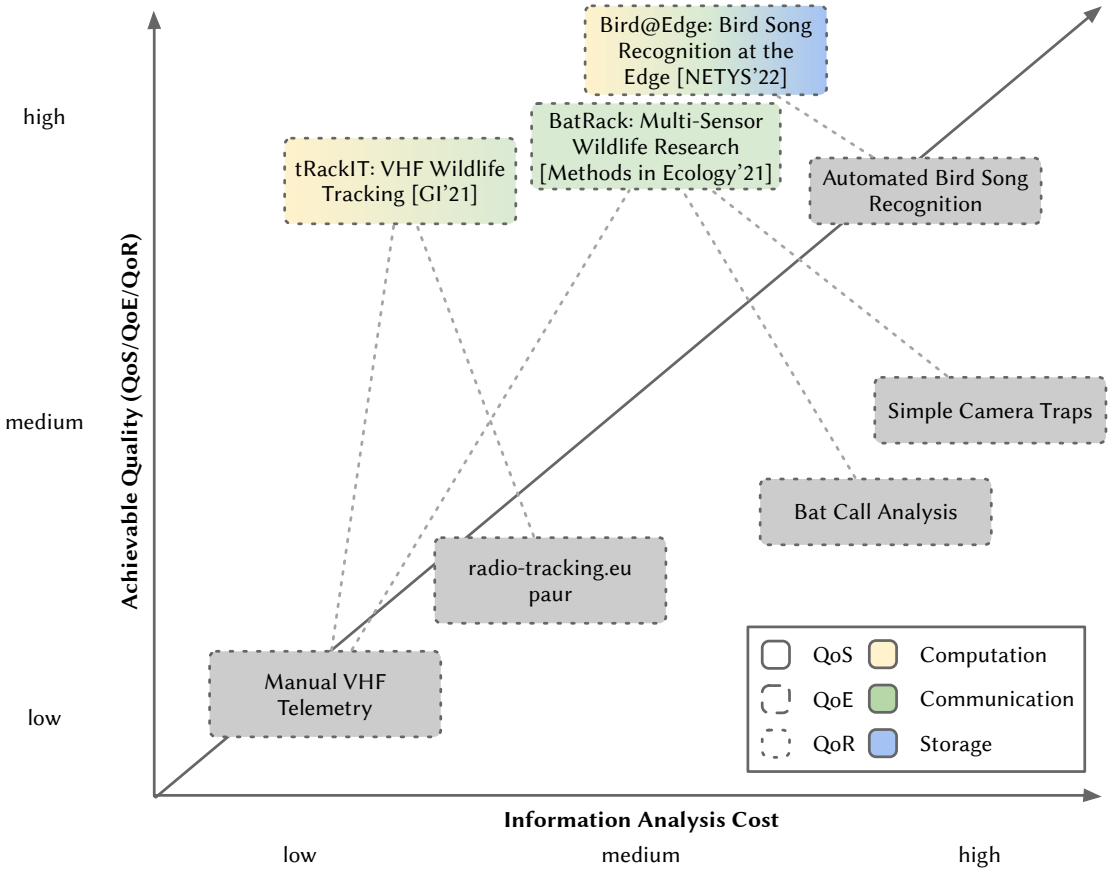


Figure 4.1: Information analysis cost and achievable quality of the contributions in the field of environmental monitoring

4.1 PIMOD: A Tool for Configuring Single-board Computer Operating System Images

4.1.1 Introduction

When applying technology in environmental monitoring or in the humanitarian field, it is particularly important that the equipment used is available and that the installations are traceable and maintainable by the user groups. For this reason and for reasons of low cost, single-board computers (SBCs), such as the Raspberry Pi, are often used as the basis for research and especially for practical applications. Various use cases have been posted, where such devices are key enablers for the proposed solutions, be it technical or general education [Sri+13; YY17], monitoring of technology [Tru+19] or monitoring in the health care sector [KR16], or various communication technologies [QA18; Bau+19; Bau+18].

When dealing with single-board computers, either for software development or when deploying hardware based on these boards, there is a lack of support for creating operating system (OS) images. There are several cases in which readily configured images and use case specific

distributions need to be distributed to users or operators. Devices like a Raspberry Pi are used at home, in applications for multimedia centers or smart homes, but also in challenging applications such as emergency response, environmental monitoring, Internet-of-Things (IoT), and smart city infrastructures.

Single-board platforms that do rely on an operating system regularly use images provided by vendors or third parties. Typically, an image is flashed to an SD card and then booted in a system. Since there is no installation process, the OSes heavily depend on defaults, e.g., username, password, installed software, or on scripts executed on the first boot, e.g., cryptographic parameters or partition size adjustments. Software can then be installed and configurations can be adapted in the running system. While this seems to be convenient for single deployments and fast progress compared classical installation routines, it is not suitable for larger deployments.

When custom software and additional configurations need to be added to an OS image, this can either be achieved by a) creating an OS image from scratch, b) adding scripts to be run on the first boot, or c) create an image from a previously configured system. However, these methods each have their individual drawbacks. Bootstrapping images from scratch requires deep technical understanding. When using first boot scripts, a network connection is required on this first boot. Creating an image from a configured system requires additional steps to revert OS specific first boot configurations.

In this section, PIMOD, a tool for modifying an existing operating system image by executing commands described in a configuration file, is presented. In the proposed line-based configuration, a `Pifile`, i.e., a small set of commands, can be used to describe how an image will be created. These commands are then interpreted by PIMOD and executed accordingly. In our approach, the target image is based on an existing image, it then can be resized, and files from the host system can be included in the image. The special `RUN` command allows running commands inside the image, so that guest OS specific packet managers and configuration tools can be used. Our approach can easily be used with continuous integration (CI) systems and enable reproducible builds of single-board computer operating system images. The software as described in this section is released under the free and open source GPL-3.0 license[5] and is available online[6].

To summarize, we make the following contributions:

- We propose a novel method of configuring single-board computer operating system images.
- We present a simple yet comprehensive operating system image configuration language.
- We provide a free and open source implementation of PIMOD.
- We conduct an evaluation of PIMOD in terms of user benefits, performance, and language flexibility.

Parts of this section have been published in Jonas Höchst, Alvar Penning, Patrick Lampe, and Bernd Freisleben. “PIMOD: A Tool for Configuring Single-Board Computer Operating System

[5] `https://www.gnu.org/licenses/gpl-3.0.html`
[6] `https://github.com/nature40/pimod/`

Images." in: *2020 IEEE Global Humanitarian Technology Conference (GHTC 2020)*. Seattle, USA, Oct. 2020, pp. 1–8. DOI: 10.1109/GHTC46280.2020.9342928.

4.1.2 Related Work

In general, single-board computers (SBCs) use operating system images provided by a) the hardware vendor, b) a third party operating system distributor, or c) a software provider, bundling up its software and dependencies to create a software-specific distribution. The images are flashed to a SD card and then booted on a SBC. Since there is no installation process, the OSes heavily depend on defaults and can only be adapted by running them and changing software or configurations.

As a first option, use case specific images can be created by using the tools provided by the SBC vendor, such as pi-gen provided by the Raspberry Pi Foundation[Ras16] or alternative approaches [Kai12]. The tool is designed to create images from scratch and highly adjusted to the specific use case. The open source wireless router distribution OpenWRT features its own build system [Fai08]. This build system is created modularly, and own packages can easily be integrated into the build process. In addition to build images from scratch, the authors created an image builder, specifically targeted for OpenWRT, which installs precompiled packages to an image. However, the image builder is targeted specifically for their operating system and does not work for others. These tools often result in long execution times, since all components are installed or even cross-compiled from scratch.

The second available option consists of tools that add custom scripts to be executed on the devices itself. With PiBakery, a graphical configuration interface for Raspbian is provided, which then creates scripts that are executed on the first boot or on every boot accordingly [Fer16]. Some distributions use the first boot for configuration, e.g., ssh keys in the case of Raspbian, which need to be taken care of manually [Ras14]. These tools have the drawback that, e.g., requested software needs to be installed on every device independently, which results in multiple identical installation processes that may lead to high network overheads. The approach also lacks the possibility of being integrated into Continuous Integration (CI) build systems.

The third widely available option is to use an existing SD card with installed software and a finished configuration. While this is a straightforward approach, it can hardly be automated. To be storage efficient, the copied image, including the partition table and file system, would need to be shrunk, which requires additional tools, such as PiShrink [Bon16].

When dealing with configuration of systems, Docker is a well known virtualization system, designed for dependency management and containerization of applications [Mer14]. Docker features a simple imperative configuration language. A new image is built based on an old image and extended by copying files, altering a Docker specific configuration or running commands inside the container. When using Docker to provide and install software, it is necessary to install Docker and the corresponding software images on the live system. Therefore, this approach does not overcome the problem of multiple installations on individual devices and does not offer a full operating system image.

For configuration management tools like Ansible [HM17], Saltstack [Hos12], and Puppet [Loo11], the main concept is to have a central server that ships a configuration to every node. The node then adapts the installed system in the manner defined in the configuration. This method has the drawback that it uses more network resources because every single node has to download updates and the installer for itself. Also, the nodes have to be booted so that the first boot scripts are executed. The possibility of configuring and reconfiguring a running system is quite helpful, but we focus on the creation of full OS system images with preinstalled and configured software. Furthermore, the client part has to be installed on every single node, and the master node has to run when a new node should be configured.

4.1.3 PIMOD Design

The goal of PIMOD is to facilitate the creation of single-board computer operating system images with custom software in an easy and reproducible manner and simplify the deployment of such devices. To reach this goal, a simple yet comprehensive configuration language is provided, which is interpreted to modify a system image. The language should be manageable through versioning systems to support the overall goal of reproducibility. A generic configuration language cannot rely on distribution-specific configuration parameters and thus should provide an interface to the distribution's configuration mechanisms. With PIMOD we target Linux-based operating systems, which are widespread in several communities using SBCs [Bau16; Joh+18].

The PIMOD Language

In this section, the PIMOD language used in a Pifile is presented. To reach the goal of easy learnability, the language was inspired by the Dockerfile language, which is already widely known. A Pifile is a line-based document where each line may either contain a) an empty line that may contain white space, b) a comment indicated by a hash symbol, c) a PIMOD command written in caps followed by parameters.

FROM <source> [partition] The required source parameter declares a base image to be found in the local file system, a block device to create an image from, or an URL to be downloaded and extracted. Optionally, the partition number resized and mounted in the further process can be declared. It defaults to the second partition, since most operating systems use one boot as one system partition.

TO <destination image> When a Pifile is executed, the resulting image is written next to the Pifile and named after the respective Pifile. The image destination can be changed by running the `TO` command. When a block device is specified, the defined source is written to the respective device and further commands are executed directly on the device.

INPLACE <image> Using the `INPLACE` command, an image can be specified on which the commands are executed.

```
1  FROM 2020-05-27-raspios-buster.img 2
2  TO raspbian-buster-upgraded.img
3
4  # Increase the image by 100 MB
5  PUMP 100M
6
7  # Enable serial console using built-in configuration tool
8  RUN raspi-config nonint do_serial 0
9
10 # Upgrade the operating system image
11 RUN apt-get update
12 RUN bash -c 'DEBIAN_FRONTEND=noninteractive apt-get -y dist-upgrade'
13
14 # Install an ssh key
15 INSTALL id_rsa.pub /home/pi/.ssh/authorized_keys
```

Listing 4.1: PIMOD example 1: upgrade Raspbian and enable the serial console.

PUMP <bytes> Using the PUMP command, the image is increased by the given amount of bytes, SI prefixes such as k, M, G or, T are supported.

PATH <location> By default, the local PATH variable of the host system is used inside the guest system. With this command, it can be extended by another location.

RUN <cmd> Commands specified using the RUN command are executed inside of the operating system image. Note that the operating system of the image is not started, but the run time environment of the target system is modeled.

INSTALL [mode] <source> <destination> Installing custom files from the host system is especially useful when custom software is used, or for configuration purposes. The source parameter relates to a file in the host file system, the destination describes a path in the file system of the target system. The optional mode parameter can be useful when installing executables, e.g., cross-compiled software.

HOST <cmd> When a command is specified using the HOST command, it is executed on the local system rather than inside the image. Issuing a local command can especially be useful for preparing configuration files or cross-compiling software, which later is installed to the guest system.

In Listing 4.1, a Pifile is presented that features all commands of the PIMOD language. Line 1 defines a source image to be found in the local file system and the partition to be resized and mounted as the primary system partition. In Line 2, we declare that the file should be written

to an alternative location. Line 5, `PUMP 100M`, causes the image and the second partition to be increased by 100 mebibytes. In Line 8, a distribution-specific configuration tool is used to enable the serial console available at the target hardware. Line 11 and 12 are used to upgrade the operating system by first updating the sources of the packet manager and then running a distribution update. Note that in Line 12 an environment variable is set by running the command inside a bash shell. Finally, in Line 15, a ssh public key is copied to allow remote login.

Linux Support

The Pifile language is designed to be a simple yet comprehensive operating system image configuration language. To reach this goal, some assumptions were made during the design phase. First, to enable fast execution of Pifiles, we do not want to use full system emulation, which would result in booting the guest system kernel. This would have the disadvantage that, e.g., the first boot scripts of the distribution would be executed and other parameters would be initialized, such as cryptographic keys, as discussed in the introduction. We decided to use a QEMU-based system emulation, which allows us to execute Linux ELF binaries across multiple different instruction set architectures [Bel05]. Second, especially mounting the partitions of the image according to the distribution requires specific knowledge, which is hard to generalize. Therefore, we decided to use the file system table defined by the Filesystem Hierarchy Standard, `/etc/fstab`, which itself is used by many Linux operating systems. Third, the executed binaries are searched according to a path variable, which itself is distribution specific. In PIMOD, this variable is initialized from the host system and can be extended by using the `PATH` command in a Pifile.

Continuous Integration Support

Continuous Integration (CI) is a technique used to overcome integration problems in the development cycle during software engineering. It has been shown that continuous integration improves the productivity of project teams and boosts the integration of external contributions without a reduction in code quality [Vas+15]. When combining version management and modern CI systems, every commit of a software under development is automatically integrated into a larger context and tested. PIMOD is designed to be used in combination with CI to create software-specific operating system images in a reproducible and easy manner.

Host System Support

Another goal of our approach is extensibility, such that it can be integrated into workflows of the communities using PIMOD. Thus, the configuration language should provide options to interface the host system. One option to enable interfacing in this manner is the already described `HOST` command. In addition, users should be able to use environment variables defined in the host system and program a control flow.

4.1.4 Implementation

The target of modifying system images and executing code inside of a system image can be achieved best by using system tools. GNU/Linux ships several helpful tools for the individual tasks implemented by PIMOD. To make use of and integrate existing tools in a simple manner, PIMOD was implemented using the Bash programming language[Ram94].

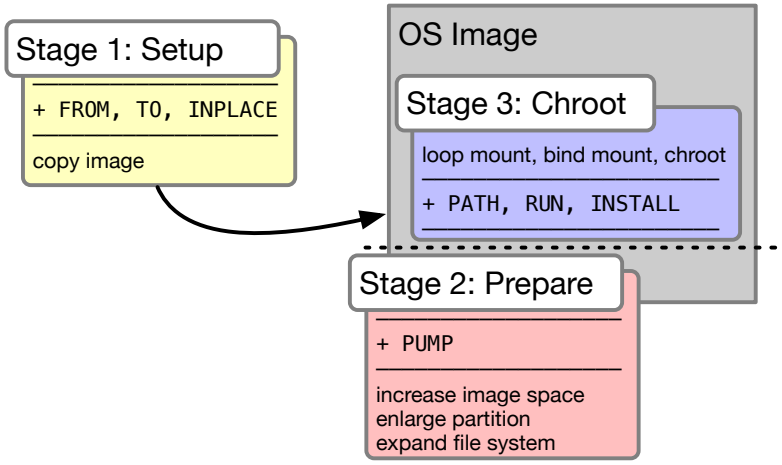


Figure 4.2: Stages of PIMOD: preparation, commands, and post-processing.

The interpretation of a Pifile is implemented in several stages, since some commands can only be executed after others, as shown in Fig. 4.2. We define three stages to which the commands are assigned: First, *setup*, managing the `FROM`, `TO` and `INPLACE` configuration. Second, *prepare*, handling changes of the image itself, currently only depicting `PUMP`. Third, *chroot*, covering all commands interacting with the file system of an image, namely `RUN`, `PATH` and `INSTALL`. During the execution of a Pifile, it is actually executed one time for each stage, only executing the commands belonging to the individual stage. This mechanism insures that source and target are defined before resizing an image, which itself needs to happen before modifying content on the guest.

Stage 1: Setup

In the setup stage, the `FROM`, `TO`, and `INPLACE` commands are executed, checking and setting source and destination system images. `FROM` is not only able to handle local images, but can also download images from a remote location by specifying an HTTP(s) or FTP URL. A local cache minimizes network load and timely overhead. When no destination is defined via `TO`, it is derived from the name of the Pifile. The stage is concluded by copying the source image to the destination location. In the special case of an identical source and destination path and when using `INPLACE`, the modifications are executed in place.

Stage 2: Prepare

The stage implements the PUMP command. Enlarging an image requires to a) increase the image file size, b) enlarge a partition inside the image, and c) expand the file systems to the size defined by the partition. These subtasks are implemented using the GNU/Linux utilities `dd`[7], `fdisk`[8], and `resize2fs`[9].

Stage 3: Chroot

Before executing the RUN and INSTALL commands implemented by this stage, some preparations need to be taken: first, the system image file is associated with a loop device of the host system. Then, the main partition's file system is mounted inside the host system. A working chroot environment requires system interaction, which can easily achieved by importing `/dev`, `/sys`, `/proc` and `/dev/pts` using a bind mount. The network interfaces are available through the host system kernel, the domain name system configuration is done by bind mounting `/etc/resolv.conf`. After this step, the statically linked QEMU binaries for the supported platforms are also bind mounted in the chroot environment. Ultimately, additional partitions defined in the file system table of the guest system are mounted. INSTALL is implemented by copying the requested files to the target file system and optionally adjusting the permissions. Running a command inside the target image is easily achievable using chroot: a command is executed in a specified root directory and thus using all binaries, libraries and resources of the mounted image.

Continuous Integration

PIMOD is designed to work in combination with continuous integration services. We provide an example integration for two different CI services. Travis CI is a free and open source CI service, which has been shown to be used by a wide variety of software projects [BGZ17]. GitHub Actions is a CI service integrated with GitHub, a software hosting platform, widely used for Open Source projects [Dai18]. In both integrations, first the dependencies need to be installed, then the resources, such as a base image are downloaded, and finally the Pifile is executed. The output of PIMOD is presented inside the web interfaces of the individual service. Also, both implementations offer the possibility to release the created image in the form of a downloadable image. Hence, the developers of a use case specific distribution can test their progress locally and tag a specific git commit. This indeed triggers a cloud build using the discussed CI integration and uploads an image to the corresponding releases web page. Our example integration is also available free and open source in a separate repository[10] and can easily be forked and adapted.

[7] `https://www.gnu.org/software/coreutils/manual/html_node/dd-invocation.html`
[8] `https://www.gnu.org/software/fdisk/`
[9] `https://linux.die.net/man/8/resize2fs`
[10] `https://github.com/nature40/pimod-ci/`

4.1.5 Experimental Evaluation

In this section, we evaluate PIMOD. First, the benefits of using PIMOD from the view of a sensor network operator are discussed. Then, the performance of the approach is evaluated by comparing execution times of exemplary commands. Third, the generalization of PIMOD is investigated by testing the software with Linux OS images of different distributions and made for different hardware.

PIMOD vs. Manual Integration

The first goal of PIMOD is to facilitate the modification of single-board computer operating system images and thereby simplify deployment. We evaluate this goal by discussing the use case of deploying nodes in a sensor network scenario. The deployment of such sensor nodes can be done in different ways. A straightforward approach would be to repeat the deployment on each node manually. First, a chosen operating system is installed, then the operator connects to each node, installs dependencies and software, and configures the system. A more complex yet more efficient way would be to install only one system by the steps presented above and clone this installation to the other systems. In some cases, the operator would need to manually alter some configurations, done during the first boot. This, however, requires the appliance to be connected to a fast Internet connection and checks on every device that the initial scripts did run correctly. The third alternative for the operator would be to build the system image by him- or herself and execute the required steps in the process. This would, however, require a deep understanding of the build process of an operating system, which can take time to understand and which itself takes a certain time to execute. In addition, some software build systems require vast amounts of resources, e.g., TensorFlow, a machine learning toolkit. Building the software requires certain tricks, e.g., swap partitions to allow for larger amounts of memory, which can be circumvented using more powerful hardware. With PIMOD, the operators can create a configuration file in which all the required steps can be described. The resulting Pifile can then be executed either locally resulting in an image to be flashed or integrated into a CI system, e.g., to build and upload an image to a certain online location. The approach can also be used to write the resulting image to a SD card, e.g., to write device-specific configuration files.

Performance Evaluation

To evaluate the performance of PIMOD, multiple example commands are executed on a single-board computer and using PIMOD. The experiments presented in Table 4.1 were executed on a Raspberry Pi 3 Model B V1.2 that consists of an ARM-Cortex-A53 with 4 cores of 1,2 GHz and 1 GB RAM. For storage, a Samsung EVO Plus 32 GB microSD of Ultra High Speed (UHS) class U1 was used, which allows read speeds of 95 MB/s and write speeds of 20 MB/s. The experiments of PIMOD were conducted on a x86_64-based server featuring two Intel Xeon E5-2698 CPUs, a total of 256 GB RAM. Storage was realized using a NVMe-based Intel SSDPEKKW512G7, with 512 GB storage and a read and write speed of up to 1775 MB/s and 560 MB/s. All tests were repeated 5 times and averaged. Although the systems themselves are quite different in

Operating System	Command	t_{RasPi}	t_{PIMOD}	Overhead
OpenWRT	`opkg update`	2.86 s	3.11 s	8.74 %
	`opkg install python`	19.77 s	32.53 s	64.54 %
	`wget -O /dev/null http://host/100m.bin`	8.92 s	0.94 s	-90.51 %
	`wget -O /dev/null https://host/100m.bin`	9.29 s	10.55 s	13.56 %
Raspbian	`apt-get update`	19.61 s	14.61 s	-25.49 %
	`apt-get install -y python3`	60.73 s	56.38 s	-7.16 %
	`dd if=/dev/urandom of=/dev/null`	2.63 s	0.50 s	-80.99 %
	`dd if=/dev/urandom of=100m.bin`	7.19 s	0.63 s	-91.24 %
	`openssl enc -aes-256-cbc`	5.52 s	3.94 s	-28.62 %

Table 4.1: Example executions times of different commands using a Raspberry Pi compared to PIMOD.

terms of performance, we try to mimic a build server that might be running in a continuous integration pipeline.

In Table 4.1, the runtimes of various commands on different operating systems are presented, namely OpenWRT and Raspbian Buster. With `opkg update` and `apt-get update`, the individual packet managers update the list of available packages. The OpenWRT packet manager introduces a small overhead using PIMOD compared to native execution. Updating the package list consists of downloading the compressed lists and verifying the signature, especially the latter requires many ARM instructions to be simulated on the x86_64 hardware and therefore introduces some overhead. Installing software using `opkg` is faster on the native hardware compared to PIMOD, which is mostly related to the used `gzip` compression executed through `qemu`.

When looking at Raspbian Buster's packet manager `apt`, PIMOD achieves better runtimes compared to native execution, for both updating the package. One reason for this is the much faster write speed of the desktop system compared to the microSD card of the Raspberry Pi. Since OpenWRT is designed for low footprint systems, such as routers, the packages are relatively small compared to the respective Raspbian packages, based on Debian and designed for desktop class computers. For the `wget` examples, the downloaded data is not written to disk, and thus only shows the performance of the networking stack. Downloading the file using HTTP shows the superiority of the build server, only requiring a tenth of the time compared to native execution. Since most build servers do have larger than 1 GBit/s Internet connections, the runtimes can be up to 10 times as fast compared to a Raspberry Pi limited to 100 MBit/s. However, when using the same protocol with TLS encryption, the cost of encryption and

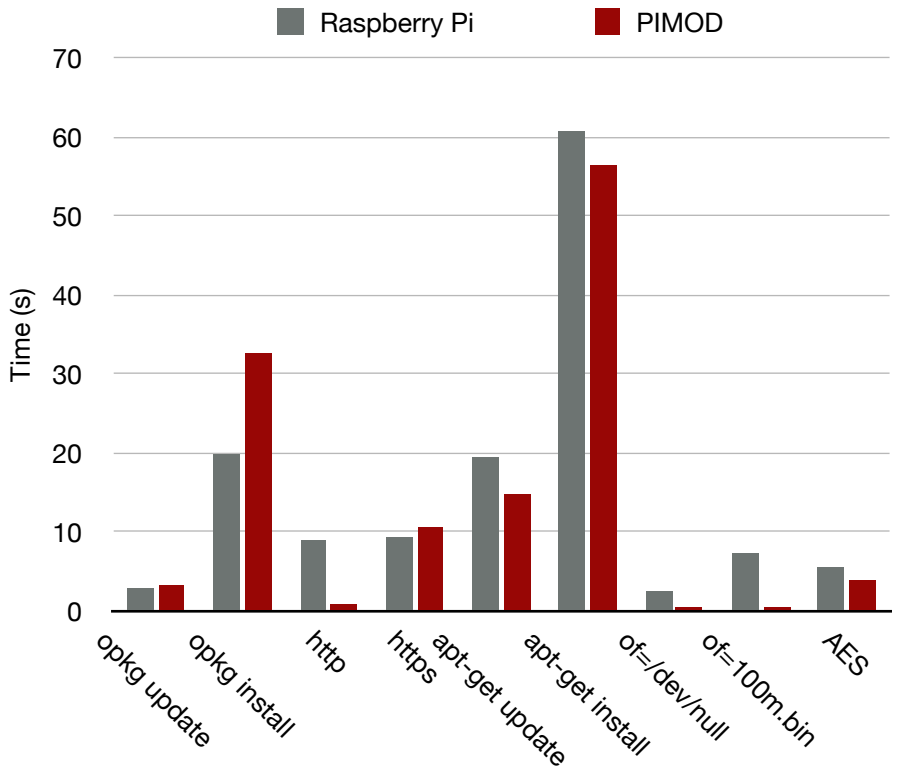


Figure 4.3: Example executions times of different commands using a Raspberry Pi compared to PIMOD.

decryption adds an overhead of 13.6%. In the examples using `dd`, 100 MB of random data is read using the command `dd if=/dev/urandom bs=1M count=100`. The computations needed to generate the data happen in the kernel and thus natively on both systems. When discarding the data using `of=/dev/null`, no heavy computation is required and PIMOD performs better, because the main load happens in the kernel of the host. From the runtimes of the second example presented in the table it becomes evident that for read/write intensive tasks, the disk is the bottleneck; the overhead gain of 91.24% is comparable to the download task. In the last example, data is read from `/dev/urandom` and encrypted using the OpenSSL AES encryption. The speed advantage of PIMOD fades, since the computationally intensive encryption is performed through the QEMU emulation layer. Nevertheless, our approach is still around 30% faster compared to native execution.

Testing Linux Distributions

To test the generalization properties of our approach, two directions are important: hardware and operating system variety. In our tests using the well known Raspberry Pi, we were able to use PIMOD for the widespread Debian-based operating systems, such as Raspbian and Ubuntu Server, as well as OpenWRT, CentOS, Fedora, Kali, and OpenSUSE. Android as well as NixOS did not implement the Filesystem Hierarchy Standard and therefore do not satisfy our assumptions. PIMOD can be adapted to work with both distributions, but a generalization is

not easily feasible. In addition, we evaluated different hardware platforms using the operating system recommended and distributed by the vendor. In our tests, we found that PIMOD can be used with the Libre Computer boards ALL-H3-CC, AML-S805X-AC and ROC-RK3328-CC, BananaPi M4, Nvidia Jetson Nano (AI development board), ODROID C2 and N2, OrangePi 3, all models of RaspberryPi, and the RockPi 4. Most of the operating systems distributed by the vendors are Debian-based and include specific changes for the individual boards.

PIMOD Language Flexibility

One secondary design goal was to support flexibility for the users of PIMOD. The presented tool is written in the Bash scripting language, a Pifile itself is a script sourced in the individual stages. Thus, in a Pifile all features available in Bash scripts can be used. PIMOD's flexibility features will be discussed by the examples presented in Listing 4.2, the respective output log is shown in Listing 4.3.

```
1  FROM http://downloads.openwrt.org/releases/18.06.5/targets/brcm2708/
       bcm2710/openwrt-18.06.5-brcm2708-bcm2710-rpi-3-ext4-factory.img.gz
2
3  # Derive block device from environment
4  TO $DEVICE
5
6  # Include wifi configuration
7  source modules/wifi.Pifile
8
9  # Add local public ssh key
10 RUN tee -a /etc/dropbear/authorized_keys <$HOME/.ssh/id_rsa.pub
11
12 # Set DHCP client mode for eth0
13 RUN tee -a /etc/config/network <<EOF
14 config interface 'lan'
15         option type 'bridge'
16         option ifname 'eth0'
17         option proto 'dhcp'
18 EOF
19
20 # Cross-compile local software
21 HOST GOOS=linux GOARCH=arm GOARM=5 go build -o dtn7d ./dtn7-go/cmd/
       dtnd
22 INSTALL 755 dtn7d /usr/bin/dtn7d
```

Listing 4.2: PIMOD example 2: advanced scripting with Bash features.

```
1  ### FROM http://downloads.openwrt.org/releases/18.06.5/targets/brcm2708/bcm2710/openwrt-18.06.5-
       brcm2708-bcm2710-rpi-3-ext4-factory.img.gz
2  Using cache: /var/cache/pimod/downloads.openwrt.org/releases/18.06.5/targets/brcm2708/bcm2710/
       openwrt-18.06.5-brcm2708-bcm2710-rpi-3-ext4-factory.img.gz
3  ### TO sd.img
4  Moving temporary /tmp/tmp.sv3nxlM9qS to sd.img
5  add map loop0p1 (253:0) : 0 40960 linear 7:0 8192
6  add map loop0p2 (253:1) : 0 524288 linear 7:0 57344
7  ### RUN tee -a /etc/dropbear/authorized_keys
8  ssh-rsa AAAA.... hoechst@ds
9  ### RUN tee -a /etc/config/network
10 config interface 'lan'
11         option type 'bridge'
12         option ifname 'eth0'
13         option proto 'dhcp'
14 ### HOST pushd dtn7-go
15 /storage/hoechst/pimod-example/dtn7-go /storage/hoechst/pimod-example
16 ### HOST go build -o dtn7d ./cmd/dtnd
17 ### INSTALL 755 dtn7d /usr/bin/dtn7d
18 ### HOST popd
19 /storage/hoechst/pimod-example
20 umount: /tmp/tmp.ampfPoR8rz/dev/pts unmounted
21 umount: /tmp/tmp.ampfPoR8rz/dev unmounted
22 umount: /tmp/tmp.ampfPoR8rz/sys unmounted
23 umount: /tmp/tmp.ampfPoR8rz/proc unmounted
24 umount: /tmp/tmp.ampfPoR8rz unmounted
25 del devmap : loop0p2
26 del devmap : loop0p1
```

Listing 4.3: PIMOD example 2 (Listing 4.2) execution log.

Environment Variables

Environment variables can be helpful to adjust the build to the runtime. In the example presented in Line 5 of Listing 4.2, a block device to work on can be specified through the environment variable `DEVICE`. In Line 11, another example is shown, where the ssh public key from the local user is added to the guest system.

Redirections

Redirections of input and output streams are a key feature in shell programming and can be helpful in our use cases. The task of adding an ssh key from the host system, presented in Line 11, is implemented by appending to a file and redirecting the input stream to a file in the host system. The `HOME` variable provides the location of the host system user's home directory. A second example is presented in Lines 14–19, where a network configuration file is written by using a *here document*. When using documents, the configuration resides inside of the Pifile itself, and is easily understandable. Environment variables are also evaluated inside of *here documents*, such that device-specific configuration, e.g., a hostname, can be set using this mechanism.

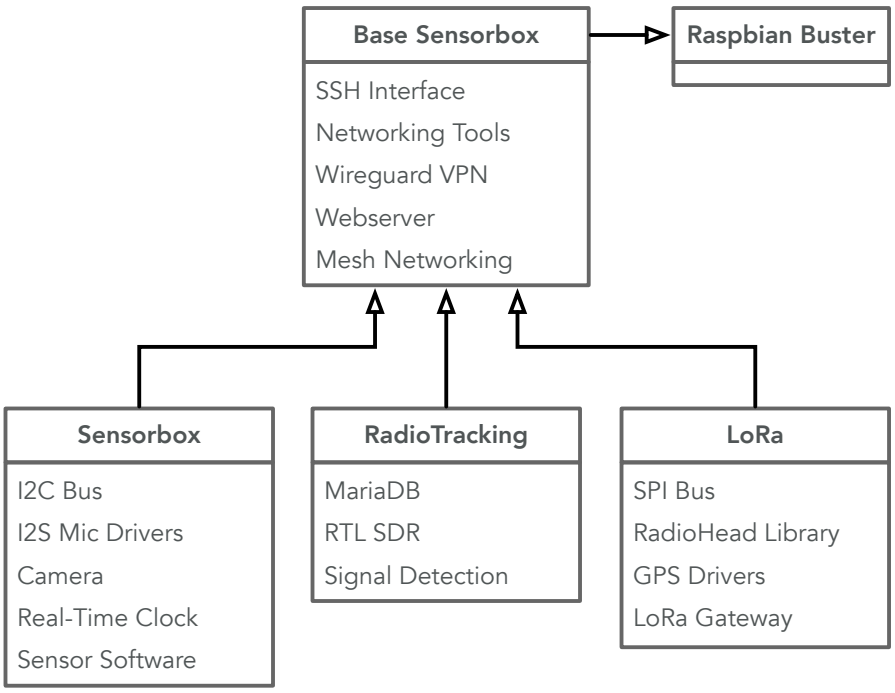


Figure 4.4: Raspberry Pi Image configurations used for the Nature 4.0 Project

Cross-Compilation

For some cases, it is beneficial to compile software on the host system, rather than running the virtualized compiler of the guest system, especially for larger software projects. In the above example of Line 22, the delay-tolerant routing software DTN7 [Pen+19] is cross-compiled on the host system by using the `HOST` command and installed to the guest system afterwards.

Modularization

Individual parts of a Pifile can be extracted into modules, represented as independent files, that can be reused afterwards. In the example presented above, the Wi-Fi configuration is included from another Pifile, as presented in Line 8. The file is included by using the `source` command, which executes the commands from the file in the current script. Thus, PIMOD offers a wide variety of interactions with the host systems and can thereby be integrated into user's workflows well.

Applications of PIMOD

The goal of the Nature 4.0 project is to develop a modular environmental monitoring system for high-resolution observation of species, habitats, and processes relevant to nature conservation [Fri+19]. For this use case, a set of sensor boxes has been designed, each dedicated to sightly different use cases, following the same principal networking functionalities and the same software basis, each based on a Raspberry Pi. In Fig. 4.4, the hierarchy derived from the

requirements of the project are presented. The first box is the rather generic *Sensorbox*, which comes in different configurations, distinguished by the wired sensor set. It includes a camera, a microphone via the I2S bus, sensors for temperature, humidity, atmospheric pressure, air quality, spectral brightness, and various others. The second box is the *RadioTracking* configuration. This box is dedicated to read and analyze data flows of four software-defined radio devices per box, which point to the four directions of the compass in order to record signals from bats tagged with special senders [Got+19]. The last configuration is a *LoRa* gateway receiving and forwarding messages of other sensor applications via the well known long range *LoRa* protocol. All images are derived from the *Raspbian* OS, in its *Buster* version, which is the de facto standard for Raspberry Pi SBCs. The *Base Sensorbox* image contains all shared software and configuration, e.g., an access option via a VPN and a SSH remote shell, a web server for convenient data retrieval and management in the field, as well as a mesh network configuration for sensor box interconnection.

PIMOD is used in the project Nature 4.0 for over a year at the date of writing, and integrates well into the workflow[11]. A software or system developer can get a copy of the project using the git versioning system, including all sub modules, and can build am image from scratch in the matter of minutes. Additional software as well as adjustments can be implemented and a new image can be built and tested locally. After a successful test, the changes are committed, and a new release is built and using the CI integration mentioned above.

4.1.6 Summary

In this section, PIMOD, a tool for configuring single-board computer operating system images, was presented. A simple yet comprehensive language for configuring these images was proposed, which offers an interface to operating system specific configuration tools. The implementation of the proposed language was presented and the staged execution of a Pifile was explained. PIMOD was evaluated by first discussing the benefits for operators of a sensor network. The performance of the proposed approach was evaluated by comparing runtimes of exemplary commands providing an overview of the overhead. Furthermore, it was demonstrated that PIMOD supports a wide variety of hardware platforms and operating systems.

4.2 *tRackIT OS*: Open-source Software for Reliable VHF Wildlife Tracking

4.2.1 Introduction

In an increasingly densely populated and anthropogenically dominated environment, a scientific analysis of the consequences of human-wildlife interaction is essential for developing evidence-based guidelines for conservation [KA20]. Understanding the impact of altered habitats on the spatial distribution of species [Saw+09], the effects of human infrastructures such as roads [Hot+15; Asc+19], and reasons for increased mortality of endangered species [Lee+19] is crucial for preserving biodiversity in a crowded world. Movement data of animals generated by recent

[11] `https://github.com/nature40/sensorboxes-images/`

technological advances support more detailed forms of analysis and insights into the behavior and ecology of threatened species than ever before [Wyc+18; Cag+10; Wal+18].

Wildlife observations can be realized with a variety of technologies. For example, GPS technology can be used to equip animals and record their movements independently of other communication infrastructures. However, size, weight, and battery life constraints prevent the use of GPS for most European songbirds and bats.

Manual radio telemetry is another option for observing small animals. However, it is extremely labor-intensive, limited to a small number of individuals that can be tracked simultaneously [Coh99], and results in spatial and temporal data with poor resolution, which might not be sufficient for meaningful scientific analyses [Mon+10].

Automated radio telemetry systems can minimize many of these disadvantages [Kay+11; Wei+16]. In previous work, some of us presented a system based on commodity-off-the-shelf (COTS) hardware for automatic radio tracking of small animals based on Very High Frequency (VHF) tags [Got+19] as part of the open source project *radio-tracking.eu*[12]. However, three seasons of long-term stationary operating time of the system in the *Marburg Open Forest* (i.e., the teaching and research forest of the University of Marburg, Germany) revealed several deficits, such as the lack of failure handling, inadequate interfaces for data transmission and health-state monitoring, and problems with time synchronization of received signals between receivers of the same station and among different stations.

In this section, *tRackIT OS*, an open-source operating system distribution for reliable VHF radio tracking of small animals, is presented. *tRackIT OS* runs on a *tRackIT station*; its basic hardware design is due to Gottwald et al. [Got+19]. We developed *tRackIT OS* to provide new software functionality according to our experiences in studying the movement ecology of both diurnal and nocturnal wildlife with a network of 15 *tRackIT stations* in densely forested terrain. In particular, we present:

- a novel approach for automated signal detection of VHF radio tracking tags,
- means to provide reliable operation of *tRackIT stations* under harsh conditions,
- efficient live data transmission for monitoring data and detected signals,
- a novel web-based user interface for intuitive configuration of *tRackIT stations*,
- a comparative evaluation of *tRackIT OS* compared to the state-of-the-art.

Parts of this section have been published in Jonas Höchst, Jannis Gottwald, Patrick Lampe, Julian Zobel, Thomas Nauss, Ralf Steinmetz, and Bernd Freisleben. “tRackIT OS: Open-source Software for Reliable VHF Wildlife Tracking.” in: *51. Jahrestagung der Gesellschaft für Informatik INFORMATIK 2021, Berlin, Germany*. LNI. GI, Sept. 2021. DOI: 10.18420/informatik2021-035.

[12] `https://radio-tracking.eu`

4.2.2 Related Work

Ripperger et al. present a comprehensive overview of existing systems for localizing small animals using different technologies [Rip+20]. The most recent projects on automated VHF transmitter tracking are ARTS [Kay+11], Atlas [Wei+16], and Motus (also called SensorGnome) [Tay+17].

ARTS consisted of towers with a height of 40 meters and top-mounted antenna arrays [Kay+11], but the system was taken down in 2010 and replaced by camera traps and GPS transmitters. ARTS was able to determine the position of a tagged individual by triangulation with an spatial accuracy of 50 meters, but rotating through channels with different frequencies reduces the time span in which each individual can be observed. *tRackIT* supports more detailed observations of movements using a higher number of stations at lower cost and less effort in construction.

The Atlas project achieves great spatial accuracy by using the *time difference of arrival* (TOA) method for direction estimates as seen from the receiver, while costs for the developed tags are low [Wei+16]. However, implementation of the receiving stations is quite expensive, a fact that probably explains why the system is only deployed in three areas in the Netherlands, England, and Northern Israel. *tRackIT* achieves comparable results with stations built from commodity off the shelf hardware at a lower price point.

Motus[13] is a globally operating network of VHF receiver stations hosted by different collaborators and supporting researchers [Tay+17]. Despite its open source character, an implementation of Motus at US$ 3000 for a single SensorGnome[14] receiver with three 9-element Yagi antennas, and US$ 7500 for a Lotek SRX800 receiver station with four 9-element Yagi antennas is costly [LN18], leading to a trade-off between spatial resolution and coverage. By default, the implemented radio receiver listens at a single center frequency and can detect pulses from tags in a narrow band of $\pm$24 kHz around its center frequency. This limits the number of distinguishable frequencies, i.e., the number of detectable individuals, substantially. Motus has delivered great insights into the ecology of different species in more than 120 research projects [Tay+17], but investigating fine-grained spatial movements by triangulation is not supported by the system. The wide frequency band that can be used by *tRackIT* supports both fine-grained temporal resolutions and observations of many individuals.

4.2.3 *tRackIT OS*

A *tRackIT* system consists of (a) VHF radio tags mounted on animals, (b) *tRackIT stations* for receiving signals emitted by VHF tags, (c) *tRackIT OS* running on *tRackIT stations* for detecting and matching signals received on multiple antennas, (d) *tRackIT servers* for collecting and presenting data transmitted from *tRackIT stations*, and (e) *tRackIT analytics modules* for deriving ecological knowledge from the collected data.

In this section, we present design and implementation issues of *tRackIT OS*, the operating system distribution for *tRackIT stations*.

[13] Motus Wildlife Tracking System: `https://motus.org`
[14] SensorGnome Project: `https://sensorgnome.org`

Requirements

Our experiences from three seasons of field work have indicated that automatic telemetry can only be a useful substitute of its manual counterpart if certain requirements are met:

1. *Low entry barrier*. To make automatic radio telemetry accessible to the widest possible user community, both hardware and software as well as data processing and analysis must be conveniently accessible, easy to use, and inexpensive.
2. *Reliability*. The used equipment must reliably record signals originating from VHF transmitters and minimize the amount of interference. Any component failures caused by adverse conditions, such as unstable power supplies, fluctuating temperatures, and hardware-based failures should be detected and handled automatically.
3. *Data availability*. In many application areas, like mortality studies [Hec20], fast data availability is highly important. Thus, direct data transmission from the field with the shortest possible delay between recording and transmission is desirable.

tRackIT Station

To deploy an operational installation in the field, a *tRackIT station* is equipped with directional antennas in the four cardinal directions, a solar panel, and a battery box. The basic hardware design is due to Gottwald et al. [Got+19]. We have slightly adapted the hardware by including an active USB hub, a better LTE modem, and a LoRa (Long Range Wireless Radio Frequency Technology[15]) expansion board (LoRa HAT), as shown in Figure 4.5.

The 'brain' of a *tRackIT station* is a Raspberry Pi 3 Model B that consists of a quad-core 1.2 GHz ARM-Cortex-A53 and 1 GB of RAM. It offers various input/output options, including Wi-Fi and 4 USB ports. The system is powered through a 5V USB port and is capable of powering connected USB devices. The four directional antennas are connected to four software-defined radios (SDR) (Nooelec NESDR SMArt v4) for signal analysis. Since these SDRs require more power than provided by the Raspberry Pi, an active 4-port USB hub (Anker 4-port Ultra Slim USB 3.0 Data Hub, A7518) is used to connect the devices. An LTE modem (Huawei E3372H) and a local prepaid data plan is used to establish a mobile Internet connection. The battery box provides a 12 V source that is converted using a step down converter rated for 2×2.4 A at 5 V. For *tRackIT stations* relying on LoRa for data publishing, a Dragino SX127X GPS HAT[16] is used. For receiving and forwarding *tRackIT stations*, the Dragino PG1301 LoRa Concentrator is used[17]. The basic hardware of a *tRackIT station* costs a total of about 200 €, consisting of 35 € for the Raspberry Pi 3B+, 4×35 € for the Nooelec SDRs, 15 € for the active USB hub, and 10 € for the power supply unit. The optional communication modules cost 50 € in the case of the Huawei LTE modem and/or 35 € (LoRa HAT) / 110 € (LoRa Concentrator) for the LoRa publish / receive upgrade.

[15] Semtech: `https://www.semtech.com/lora/`
[16] Dragino SX127X: `https://www.dragino.com/products/lora/item/106-lora-gps-hat.html`
[17] Dragino PG1301: `https://www.dragino.com/products/lora/item/149-lora-gps-hat.html`

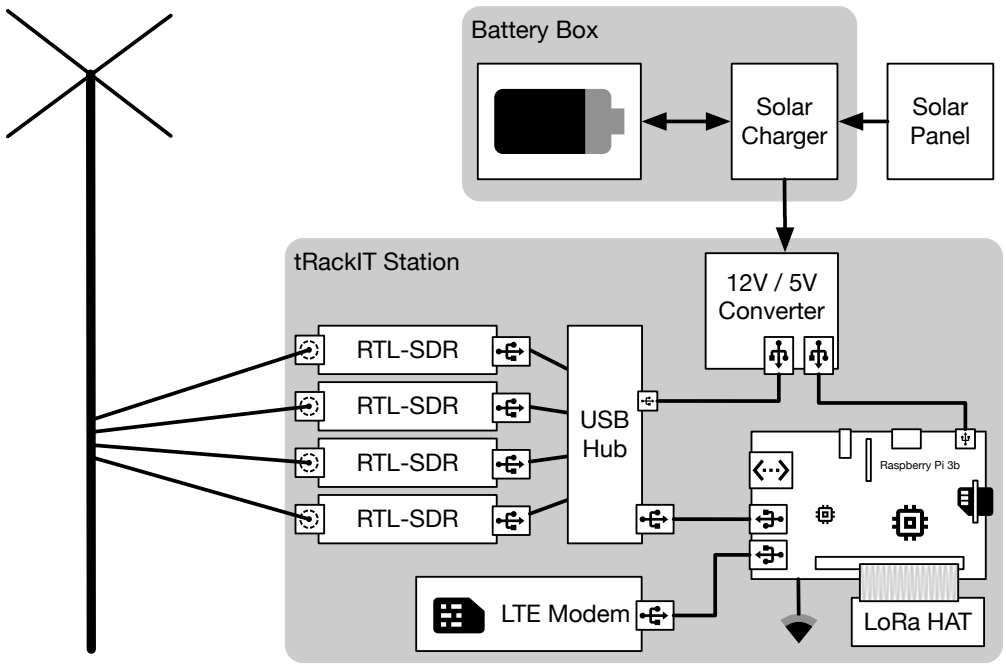


Figure 4.5: The hardware components of a *tRackIT* station.

tRackIT OS Components

The operating system (OS) plays a crucial role in the reliable autonomous operation of the presented hardware. We developed a custom distribution of the Raspberry Pi OS, called *tRackIT OS*. The primary task of *tRackIT OS* is to execute a signal detection module, called *pyradiotracking*, in a reliable manner. The secondary task is to interface with users (a) interactively while setting up the station, and (b) continuously during autonomous operation for extended monitoring. *tRackIT OS* is built using PIMOD [Höc+20b], which allows configuration of single-board computer system images in a reproducible manner. The resources required to build *tRackIT OS* as well as the OS image itself are released under a GPL 3.0 license[18].

In Figure 4.6, the main software components of *tRackIT OS* are presented. Station-initiated communication is handled using the *Message Queuing Telemetry Transport* (MQTT) protocol, with *mosquitto* as an MQTT client and server implementation [Lig17] for message distribution. It is configured such that incoming messages are forwarded to remote MQTT brokers for further processing. These brokers are also responsible for detecting and resolving connection failures.

The core software component for signal detection is called *pyradiotracking*. The component reads samples from all four SDRs, as well as detects, filters, and matches signals of VHF tags. Detected signals are saved to local storage, displayed via a custom web user interface, and published to a local message bus that is responsible for data distribution. Section 4.2.3 discusses the implementation details of *pyradiotracking*.

[18] *tRackIT OS*, available online `https://github.com/Nature40/tRackIT-OS`

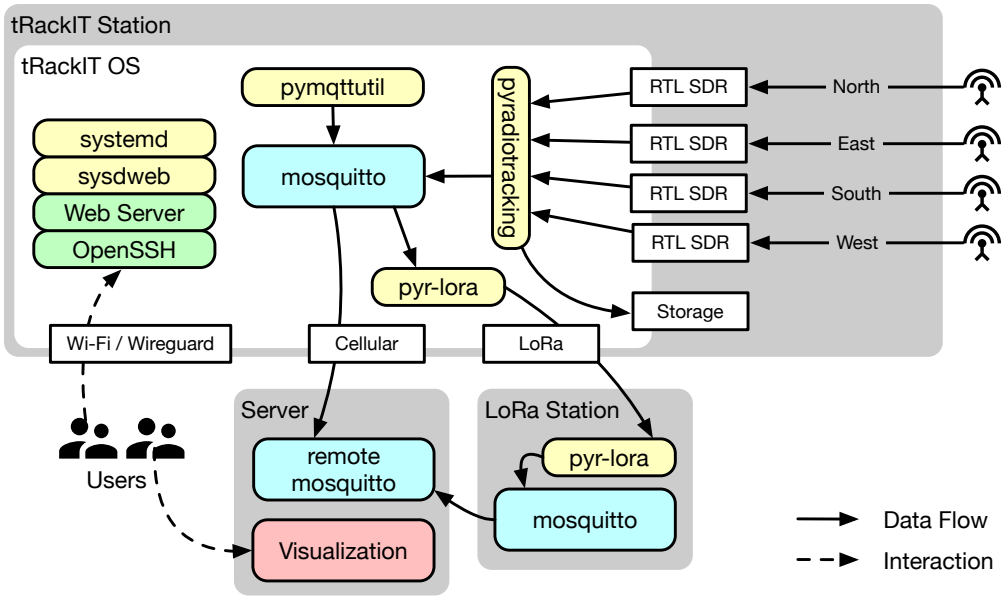


Figure 4.6: Overview of the main software components of a *tRackIT OS* distribution.

For system monitoring, we implemented a custom tool called *pymqttutil* in the Python programming language. It is released under a GPL 3.0 license[19]. The tool executes configurable Python statements in a fixed schedule and publishes the corresponding results via MQTT. It is configured such that relevant system metrics are published in a 5 minute interval, i.e., temperature, system uptime, system load, memory usage, CPU frequency, network addresses, storage utilization, and cellular data usage.

All services are managed by *systemd*. The WebUI *sysdweb*[20] for *systemd* is configured to allow easy log access and service control for (mobile) users. The *Caddy* web server is used to provide convenient access to the local storage, *pyradiotracking* and *sysdweb*. Finally, OpenSSH provides direct system access for local and remote users. To allow secure remote access, *wireguard* is used as a virtual private network (VPN).

Signal Detection

The signal detection algorithm is implemented in the *pyradiotracking* Python package, which is released under a GPL 3.0 license[21]. In Figure 4.7, the stages of signal processing are presented in a block diagram. First, spectrograms of the incoming IQ samples are created, which are used to detect signals. The detected signals are then filtered for shadow signals of lower power in neighboring frequencies and sent to a central signal queue. The detected signals of multiple antennas are matched by time and written to a local file, published to the MQTT message bus, and visualized in the local dashboard.

[19] *pymqttutil*, available online: `https://github.com/Nature40/pymqttutil`
[20] *sysdweb*, available online: `https://github.com/Nature40/sysdweb`
[21] *pyradiotracking*, available online: `https://github.com/Nature40/pyradiotracking`

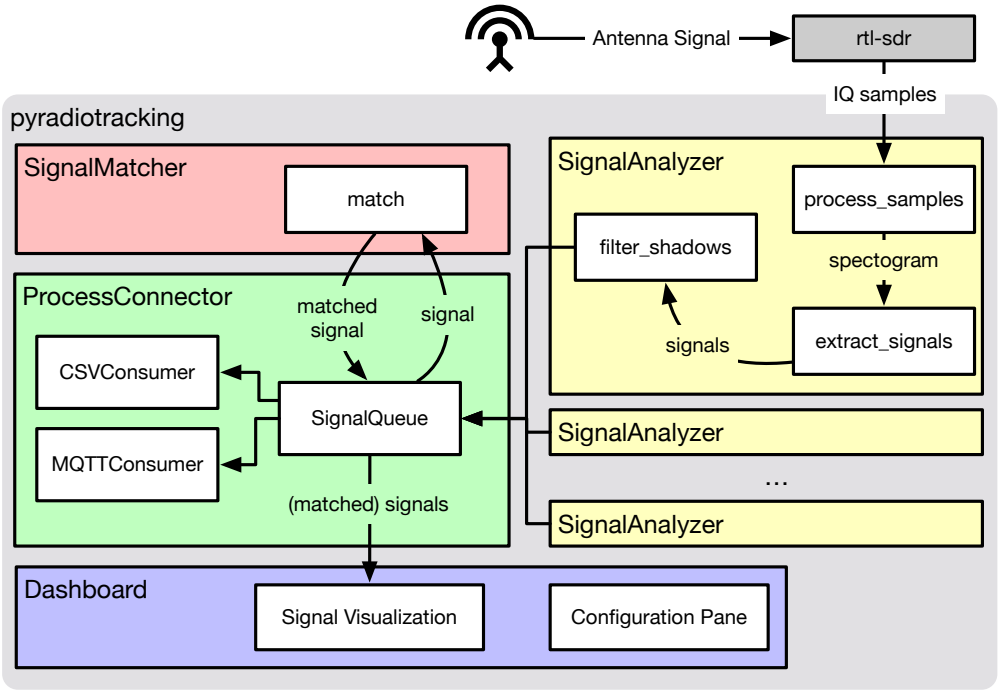


Figure 4.7: Signal analysis stages implemented in *pyradiotracking*.

To illustrate how the different stages work, data of the length of one second is used as an example. An SDR is configured such that a center frequency of 150.150 MHz, a sample rate of 300 kHz, and a fixed gain of 49.6 dB are used. A test tag with the frequency of 150.172 MHz and a signal duration of 40 ms was placed near to the receiving antenna. In Figures 4.8, 4.9, and 4.10, three stages of signal processing are visualized.

Figure 4.8 shows the raw IQ samples received by the SDR. Following the example configuration described above, there are 300,000 samples, hence 600 kilobyte of data collected in one second. In the time interval of $t_0 = 0.45\,s$ to $t_1 = 0.49\,s$, the IQ samples contain high values, which appear as a rectangle in the visualization. This rather sharp rectangle indicates that the gain value is set too high and the signal is clipping. When setting up stations for regular operation, the gain value must be chosen such that a good compromise of gain and clipping is achieved.

To detect single signals from the received data, a spectrogram is computed and processed further. This is achieved by applying consecutive Short-time Fourier Transforms (STFT) [All77] to the data. Figure 4.9 shows the spectrogram computed from the previously presented example data. The STFTs are computed with 256 samples per Fast Fourier Transform (FFT) and no overlapping samples. The Hamming window function is applied to smoothen discontinuities at the start and the end of the processed FFT. In this configuration, the bandwidth of 300 kHz is divided into 256 bands and a frequency resolution of 1,171 kHz, to achieve a time resolution of $1.0\,s/1,171 = 0.853\,ms$.

In Figure 4.10, signal detection on individual frequencies is visualized. The signal power (dBW) in the logarithmic scale is plotted for four example frequencies near the test sender's frequency, and the signal emitted by the test sender can be observed in three of those. The gray dashed

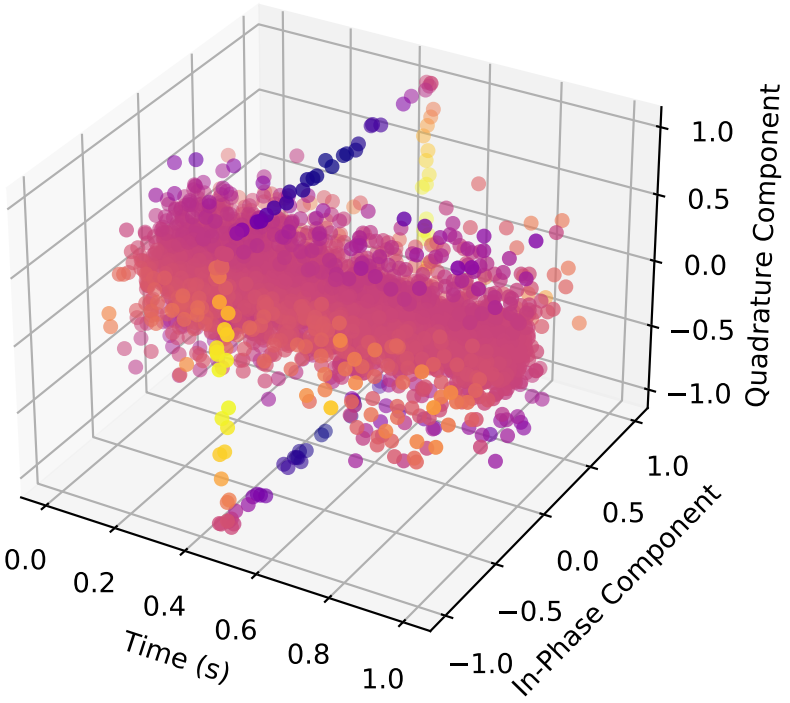


Figure 4.8: IQ samples of one second, as received by RTL-SDR.

horizontal line indicates the configured signal power threshold of -60 dBW. The gray dotted vertical lines show scan points used for initial signal detection. The blue arrow marks the total detected signal length. Signal detection is achieved by (a) iterating through all frequencies and (b) iterating through time using scan points placed according to the minimal detectable signal duration of 8 ms in our example. The signal-to-noise ratio (SNR) is calculated using the ratio of the current power and the average signal power of this frequency. If signal power and SNR at the scan points are above the configured thresholds, a potential signal is detected. The scan is then continued by evaluating the thresholds for all neighboring values until the thresholds are undershot, indicated by the blue arrows. If the duration of the detected signal is within the set limits, further complementary features are computed and added to a list for further processing.

After all signals of a spectrogram are extracted, shadow filtering is performed. We define a shadow signal as a signal that matches another signal in duration and time, but has a lower detected power. In the example of Figure 4.10, the signals detected at 150.170 MHz and 150.172 MHz would be shadow signals of the 150.171 MHz signal. The shadow signals are removed and the detected primary signals are added and written to disk, published via MQTT and sent to *pyradiotracking's* main process for signal matching and data presentation.

To improve reliability, a direct control component is introduced. The `librtlsdr` library used to retrieve data from an SDR works in such a way that as soon as requested data is available, a callback method is called. If the system load is too high and the callback method takes longer than the acquisition of the next samples, individual samples are omitted. Hardware and library-specific errors may lead to no callbacks at all. The first problem is monitored by

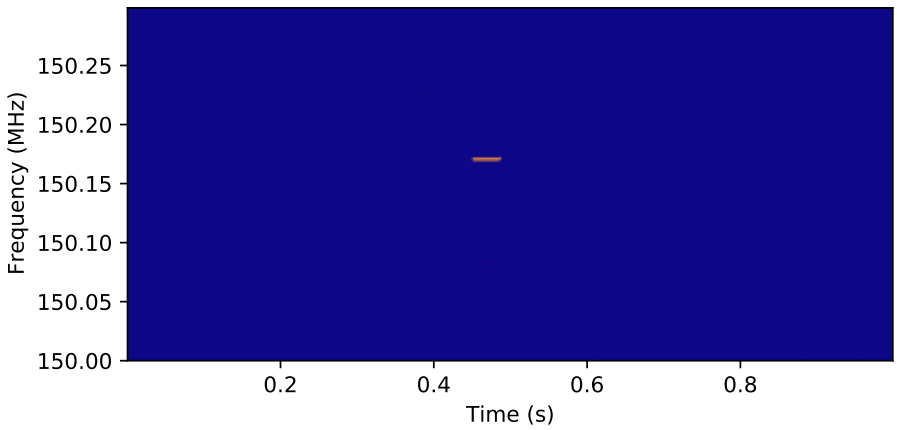


Figure 4.9: Power spectral density (PSD) of samples computed via Short-time Fourier Transform (STFT).

comparing the actual received samples with the expected number of samples using the system clock. In this way, dropped samples can be detected, even accumulated over longer periods of time. The second problem is solved by (re-)setting a periodic alarm, comparable to a dead man's switch. If the callback method is not called in time, an alarm is triggered. This terminates the analysis process, which is then restarted by *pyradiotracking's* main process.

Signal Matching

The detected and filtered signals of multiple antennas are consumed by the signal matcher, which works as follows. In a list, all currently active signal groups are held. When a new signal is detected, it is compared to each of the active signal groups in time, duration, and frequency. The SDR devices used in the project do not work synchronously and use individual quartz crystals as their clock sources, hence time and frequency mismatches are likely to happen. If all parameters of an active signal group are within the configured thresholds, the signal is added to the corresponding group. If no corresponding group is found, a new active signal group with this signal is created and added to the list. After a certain timeout, the active signal groups are removed from the list and the key features are written to disk and published.

Data Publishing

Detected signals are published directly to disk in CSV format and via MQTT in the CBOR format, which is a binary format and introduces smaller overheads compared to text-based formats. The MQTT broker running on a *tRackIT station* can be configured to forward published signals to other brokers, such as a central server via a cellular network or other IP-based networks.

In addition to this IP-based data publishing, LoRa can be used to publish signals. LoRa is a physical layer protocol based on *chirp spread spectrum* (CSS) modulation, that is robust against channel noise, multi-path fading, and the Doppler effect. This allows transmission ranges of a

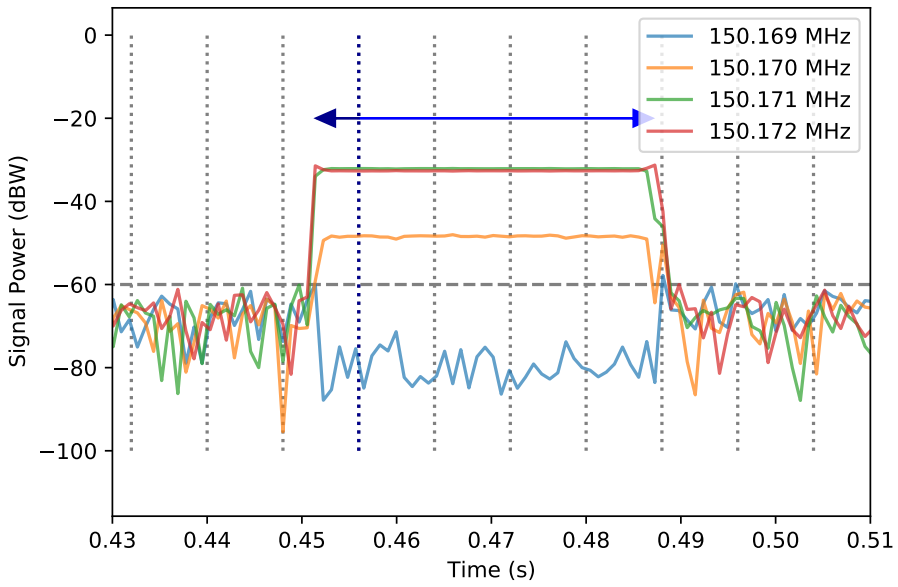


Figure 4.10: Power spectral densities (PSDs) of selected frequencies, minimal signal power threshold, and signal power sampling points.

Field	**Accuracy**	**Size (bit)**
Time	ms of current min	16
Frequency	offset to 150 MHz in kHz	9
Duration	ms	6
Signal Availability	flags	4
Signals	3-decimals	$[1-4] \times 17$
		$52-103$

Table 4.2: *tRackIT station*'s LoRa matched signal payload: fields, accuracy, and sizes.

few kilometers in urban environments and up to 15 kilometers in rural areas, with minimal power requirements but also only low data rates between 300 bps and 50 kbps [Mdh+17; Pet+17; Höc+20a].

The LoRa publishing service of *tRackIT OS* receives signals through the local MQTT broker, converts the data in a custom data-saving binary format, and sends it via LoRa. Table 4.2 shows the fields used for a matched signal's payload, including accuracy and required size in bits. A matched signal contains a minimum of one and a maximum of four signals, depending on the number of antennas that received the signal, hence the final payload size is 52 up to 103 bits. Zeros are appended to the payload to reach the required byte boundaries, resulting in messages of 7, 9, 11, and 13 bytes, depending on the number of the contained matched signals. Compared to the already compact representation in CBOR of a 4-component matched signal (56 bytes + overhead), a reduction of up to 77% is achieved (13 bytes). In the most robust LoRa settings (SF:12, BW:250 kHz, CR:4/8), such a shortened message would require 594 ms

Time-on-Air (ToA) (using Implicit Header mode with a 1-byte sender ID, the total length of the packet is 14 bytes). Following the duty cycle regulation of a maximum utilization of 1% (10%) per band, a message could be sent every 59 (5.9) seconds. While these settings do not allow continuous monitoring of individuals, sparse reporting of single observations are still of value, when trying to detect tags fallen off or with empty batteries. For stations in closer physical proximity to the receiving gateway, less robust settings may be chosen. Using a less robust LoRa setting (SF:8, BW:250 kHz, CR:4/8), the ToA drops down to 45.5 ms, allowing messages to be sent in an interval of 4.6 (0.46) seconds. From previous measurements in the Marburg Open Forest, signals could be reliably transmitted over 600 meters using this setting.

4.2.4 Experimental Evaluation

In this section, we evaluate *tRackIT OS* in benchmarking scenarios and in field experiments. The data of all experiments is publicly available at `https://github.com/Nature40/hoechst2021tRackIT-eval`.

Experimental Scenario

To evaluate *tRackIT OS* in a realistic manner, we use a system setup in the *Marburg Open Forest*, consisting of 15 *tRackIT* stations; 5 of them are used in our evaluation described below. The experiments are carried out twice: (a) with the most recent *tRackIT OS 0.7.0* and (b) using the most recent stable operating system version of the *radio-tracking.eu*[22] project [Got+19], called *paur 4.2*. We activated a test tag and carried it around in the area of the selected *tRackIT stations* together with a GPS receiver to receive ground truth data. The experiment took place over the course of 0:51:10 h with a VHF sender of 600µW power, 20 ms duration and an interval of one signal per second, which results in 3,193 sent signals. In Figure 4.11, the GPS trace of the conducted experiment is presented; stations are marked by the white circles, and the trace is colored to indicate the time component of the experiment.

Our observations using *paur* in two seasons of 2019 and 2020 indicated high numbers of falsely detected signals. We were not able to distinguish between true and false positives through the information available after signal detection. Thus, we used a power signal threshold. The *paur* experiments conducted in this section showed the same low precision, hence all detected signals with a power lower than -78 dBW were removed for further processing. Using *tRackIT OS*, this threshold is not required, since we observed very low numbers of falsely detected signals.

Signal Delay

A second observation from our previous field seasons in 2019 and 2020 is a delay in signal detection using *paur* in the order of seconds to minutes. In Figure 4.12, an example of observed signal delay is visualized. The dots show the received signal strength measured on multiple antennas of the same *tRackIT station*. Every antenna received a series of signals with low

[22] `https://radio-tracking.eu`

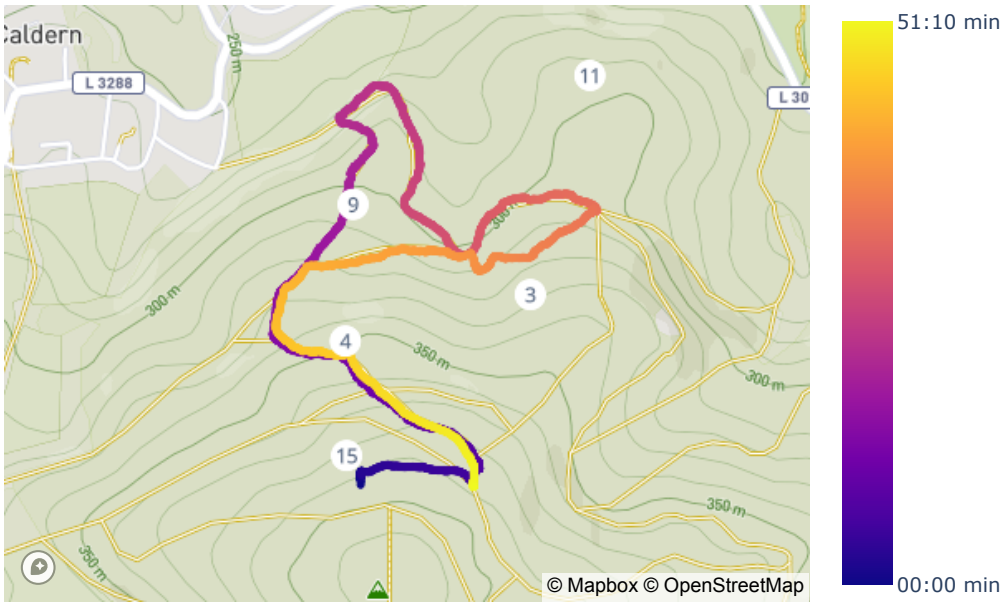


Figure 4.11: GPS trace of the experimental evaluation track and the corresponding *tRackIT* stations.

variance in signal strength that appear to be a straight line, indicating that the tag is not moving. However, these straight lines on the individual receivers are offset in time from each other, which makes further processing of the data difficult and and leads to worse to unusable bearing calculation. In the experiments of this section, we measured a delay in signal detection of 8 seconds in *paur* and no recognizable delay in *tRackIT OS*.

Signal Detection

In the *tRackIT OS* experiments, the selected five *tRackIT stations* detected 30,507 signals, each on potentially four antennas, resulting in an average of 1,525 signals (47.8%) detected per antenna. Signal detection depends on various factors, such as geographical and topographical conditions, the orientation of the antenna, the height of the transmitter above the ground, air humidity, and forest cover. Figure 4.13 shows the numbers of detected signals by station and antenna. Due to the positioning of the stations, the orientation of the antennas, and the selected test area, some of the antennas receive only a very small amount, others a large amount of the test signals. On the north antenna of station 11, only 95 (3.0%) signals could be detected, while 2,611 (81.8%) signals where detected on south antenna of station 9.

In addition to this quantitative analysis of signal detection, we evaluated the distances between the test tag and the *tRackIT stations*. Figure 4.14 shows the distance of the tag and stations measured via GPS and the power of the detected signal. While most stations can detect signals at distances of up to 400 meters, stations 4 and 11 detect signals up to 800 meters away. While the correlation of signal strength and measured distance is straightforward, a high variance can be observed from the data and signal strength alone, hence this is not a suitable estimator for distance in the presented experiment. Initially, the overall performance of the two systems

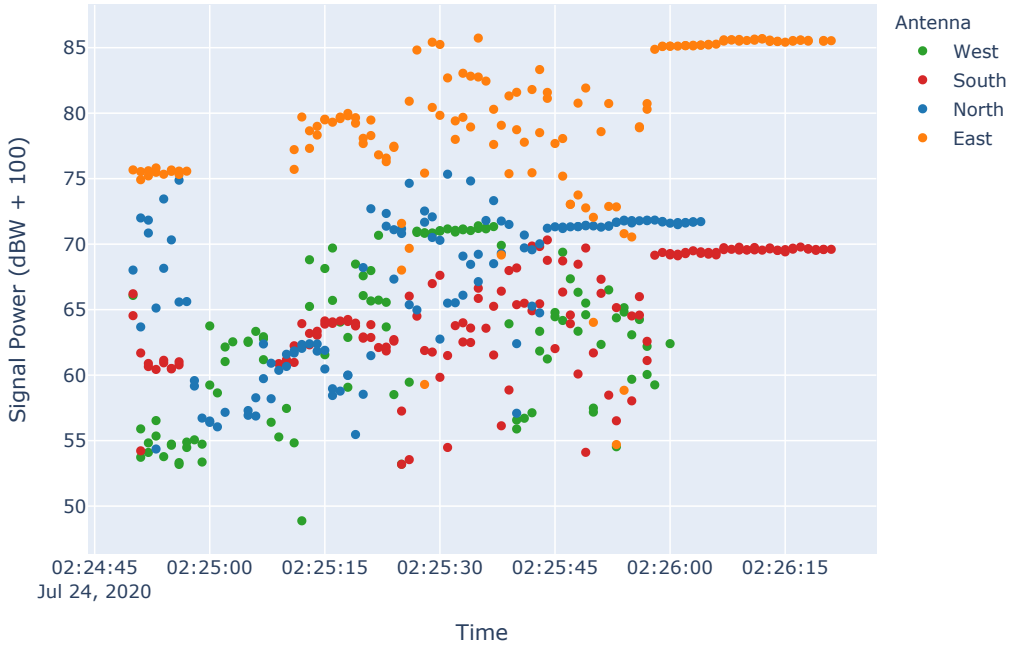


Figure 4.12: Example of signal delay among different receivers observed in the 2020 field season using *paur*.

appears comparable, especially for signals with high signal strength. While *paur* received 2,728 signals usable for bearing calculations, *tRackIT OS* received 4,438 such signals, an increase of 62.7%, when applying the same -78 dBW threshold. In addition, *tRackIT OS* received 1,108 signals of lower signal strength, which corresponds to an effective increase of 103.3% compared to *paur*.

Bearing Calculation

To reach the goal of signal triangulation, signals detected on multiple antennas of a station are used to calculate bearings. We use the method proposed by Gottwald et al. [Got+19] to produce comparable results for our bearing calculation. First, the pair of neighboring antennas with the highest and second highest signal strength are selected (s_l, s_r) and the relative gain difference δg is computed using the maximum signal strength difference δm: $g = \frac{s_l - s_r}{\delta m}$. Second, the signal strength is used to calculate the bearing between the antennas following the formula derived from the cosine theorem $\omega = \frac{\pi}{90} \times arccos(\delta g)$.

Figure 4.15 shows a histogram of bearing errors in *tRackIT OS* and *paur*. Due to an error on station 4 which was not resolved automatically, signal detection failed on this station in the *paur* experiment run, hence no data is presented in the histogram. While *tRackIT OS* has a mean bearing error of 23.7° and a standard deviation of 30.7°, *paur* not only has a lower total bearing count, but also results in 38.9° mean bearing error with 42.6° standard deviation. These results indicate that *tRackIT OS* is superior to *paur* that represents the current the state of the art in this field.

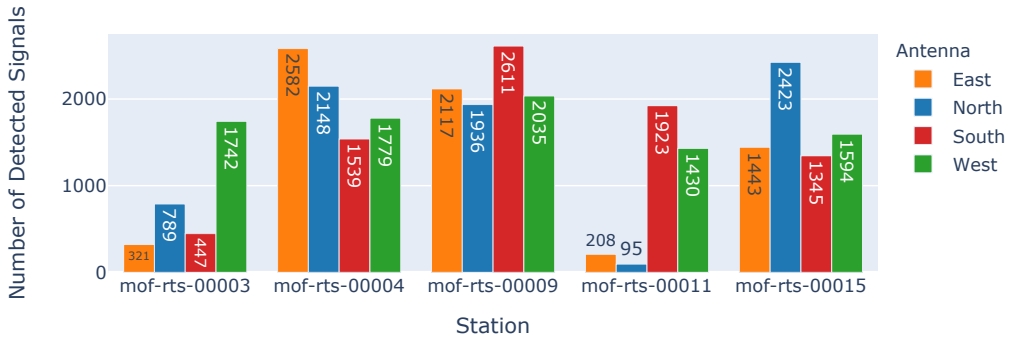
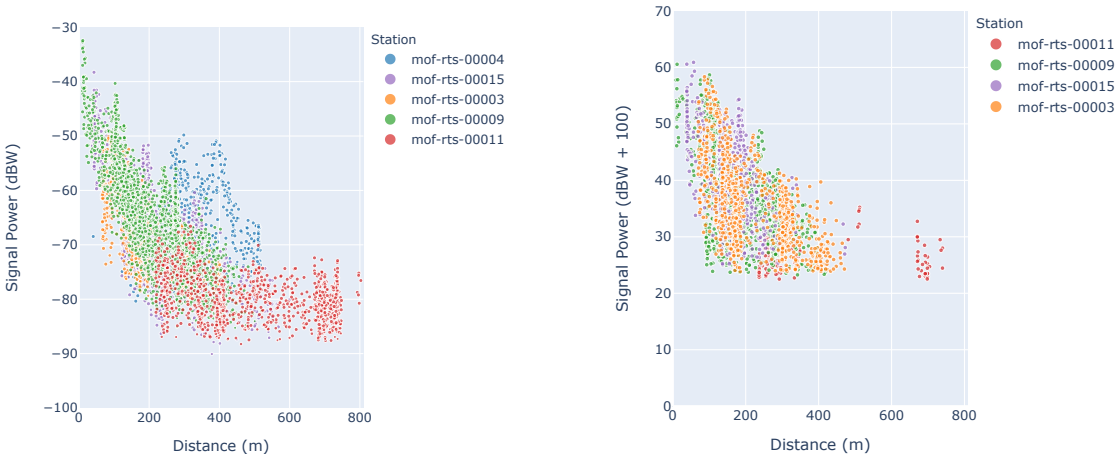


Figure 4.13: Detected signals on *tRackIT stations* in the experimental scenario.



a) tRackIT OS b) paur

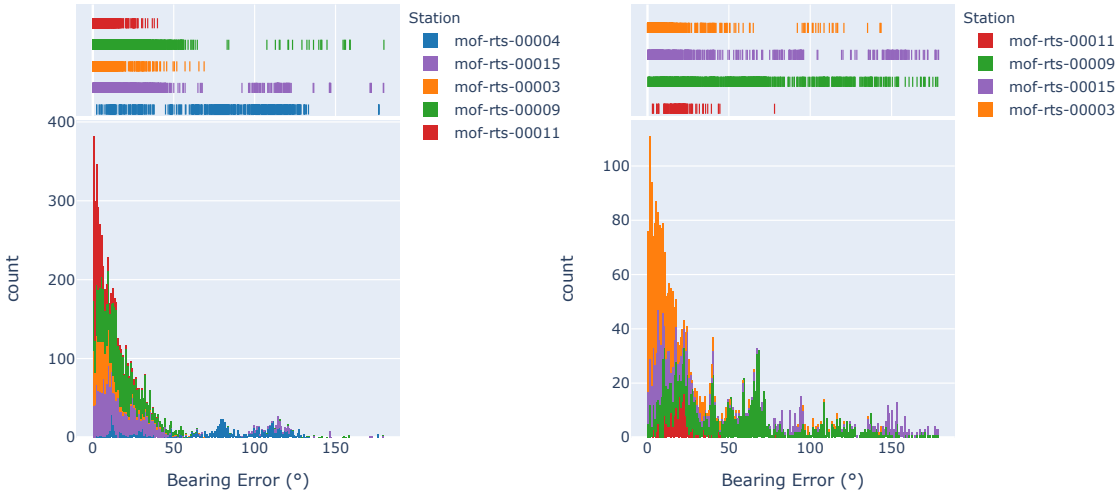
Figure 4.14: Signal power and distance to a receiving station.

Power Requirements

To operate stations autonomously and to monitor and transmit data, a stable power supply is necessary. To get realistic values for the required power, we measured a *tRackIT* station at the 12 volts input using a Monsoon High Voltage Power Monitor[23].

Figure 4.16 shows the power demands of *paur* and *tRackIT OS*. In contrast to *tRackIT OS*, *paur* does not start signal detection automatically. After all SDRs and signal analysis threads are running, *tRackIT OS* consumes an average of 8.23 W (684 mA), while *paur* consumes an average of 8.03 W (667 mA), which is an overhead of 2.55%. We also carried out experiments with varying sample rates (225 kHz – 300 kHz), but did not observe varying power demands. The systems used in the Marburg Open Forest use 12 V batteries with a capacity of 120 Ah (1440 Wh) of which only 80% should be used to limit wear, which allows a maximum theoretical runtime of 140 hours, or roughly 5.5 days. To allow a continuous operation, a 300 Watts peak solar panel is connected via a solar charger that even works during cloud cover. The presented

[23] Monsoon Solutions Inc. High Voltage Power Monitor: `https://www.msoon.com/online-store/High-Voltage-Power-Monitor-p90002590`



a) tRackIT OS b) paur

Figure 4.15: Histogram of bearing errors.

results show only a slight increase in power consumption of *tRackIT OS* compared to *paur*, i.e., *tRackIT OS* meets the power requirements for continuous operation of the system.

4.2.5 Summary

We presented *tRackIT OS*, open-source software for reliable VHF radio tracking of small animals in their wildlife habitat. *tRackIT OS* is an operating system distribution for *tRackIT stations* that receive signals emitted by VHF tags mounted on animals. *tRackIT OS* encompasses components for VHF signal processing, system monitoring, configuration management, and user access.

We evaluated and compared *tRackIT OS* against a previous operating system distribution (called *paur*), in an experimental field evaluation carried out in the *Marburg Open Forest*. Our experimental results showed that compared to *paur*, *tRackIT OS* (a) enables reliable VHF signal detection for bearing calculation, (b) increases the number of usable signals by 103.3%, (c) improves the mean bearing calculation error from 38.9° to 23.7°, and (d) introduces only a slight overhead in power consumption of 2.55% or 0.2 W. *tRackIT* has the potential to substantially improve the quality of habitat usage studies and/or environmental assessments in the context of anthropogenic interventions in the environment, while massively reducing the time required for field work.

These results show that *tRackIT OS* is a smart solution in the sense of this thesis. In Figure 4.1 on page 26, *tRackIT OS* can be classified based on the following considerations: *tRackIT OS* was able to detect 2.033 times the number of signals compared to paur, and the bearing calculation errors could be reduced from 38.9% to 23.7%, i.e., improved by a factor of 1.391. Power consumption as the main cost metric on the other hand only differs slightly, i.e., it is 1.0255 times higher than using paur. Depending on the weighting of the two quality improvements, the quality-cost improvement thus ranges from 1.356 to 1.982 of tRackIT OS compared to paur.

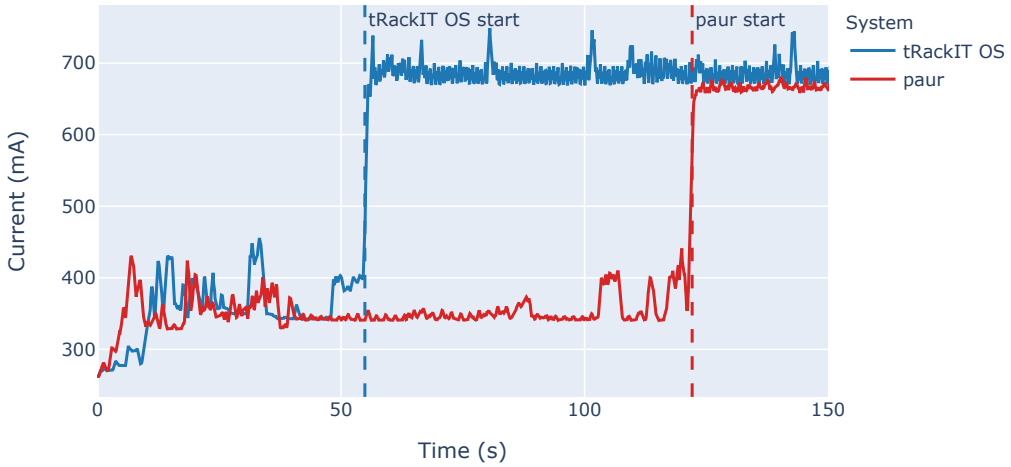


Figure 4.16: Power measurements of *tRackIT OS* and *paur* in default settings.

4.3 *BatRack*: An Open-source Multi-sensor Device for Wildlife Research

4.3.1 Introduction

Many of the important findings and principles of ecology and conservation biology have been derived from behavioural observations [CS10], but these are difficult to obtain for small wildlife [Kel08]. To further close knowledge gaps, the constraint of ecological surveys between grain and extent must be further resolved. This requires automatic, cost-effective and data-efficient (i.e. triggered) observation systems that provide comprehensive sensor combinations and enable automatic observation at the individual level.

To combine the desirable features of audio and video monitoring, BatRack was developed, a modular observation system that integrates audio, video and automatic VHF radio tracking in a single unit. The three recording technologies can be used separately, simultaneously or in mutual trigger mode, and the corresponding configuration scheduled and switched automatically. BatRack's hardware is assembled from off-the-shelf components and its design and the required software have been published under a GNU GPL 3.0 license.

In the following, BatRack's hardware and software is presented, the suitability of its audio and VHF sensors in triggering the camera is evaluated, and the potential of VHF recordings in the identification of individuals in videos, in a case study of the dawn-swarming behaviour [Kun82] of Bechstein's bat Myotis bechsteinii is tested. To date, only a few studies have examined this behaviour in detail [NK13]. Using BatRack's combined sensor approach, new insights based on individual-related information about the reproductive state and an individual's decision to change roost sites during the night can be provided.

Parts of this section have been published in Jannis Gottwald, Patrick Lampe, Jonas Höchst, Nicolas Friess, Julia Maier, Lea Leister, Betty Neumann, Tobias Richter, Bernd Freisleben, and Thomas Nauss. "BatRack: An Open-source Multi-sensor Device for Wildlife Research." in: *Methods in Ecology and Evolution* (July 2021). DOI: 10.1111/2041-210X.13672.

4.3.2 Related Work

Video recordings are often used to observe the behaviour of individuals [Car+17], but for small species camera traps are effective only over short distances [RK18]. The observation of bats is particularly difficult due to their nocturnal lifestyle in often richly structured habitats. Instead, recordings of echolocation calls are frequently used to monitor the presence/absence of bats [Mil+20]. These acoustic signals, with their comparatively long range [Ena+19], can serve as triggers for visual sensors. The combination of audio and video can additionally support the interpretation of the data [Bux+18].

However, recognizing individuals on images, particularly small and nocturnal species or species that lack unique visually detectable features, is challenging [Row+08], as is the recognition of individuals based on acoustic recordings [Sto+19]. By contrast, the automatic tracking of bats using lightweight VHF radio transmitters offers several advantages [Got+19; Kay+11; Tay+17]: (a) VHF signals can be used as triggers for other sensors and (b) they may support the recognition of individuals in video sequences based on comparisons of the VHF signal patterns with the movements observed in the video.

4.3.3 Materials and Methods

The BatRack system

BatRack combines a core computation component with three sensor units (audio, video and VHF) and tailored analysis modules. Scientists and practitioners can easily assemble, configure and extend the system. Moreover, BatRack is inexpensive (650€ without a power supply), easily repaired using commodity off-the-shelf (COTS) components (Figure 4.17), and uses free and open source software (FOSS). In addition, it is configurable with respect to the attached sensors as well as their recording ranges, time-based scheduling and mutual trigger mode. A detailed description of the hardware and software modules, including product specifications and blueprints, can be found at the BatRack webpage `https://nature40.github.io/BatRack/`.

For easy deployment, the software comes as a customized Raspberry Pi OS image bundle called BatRackOS `https://github.com/Nature40/BatRackOS/releases/`, which was built using PIMOD [Höc+20b].

The audio module (Figure 4.18a) is implemented using *pyaudio* for audio data retrieval and *numpy* for further audio processing and bat call detection. The sampling rate depends on the hardware (e.g. 384 kHz for Dodotronic Ultramic 384k). The camera analysis module (Figure 4.18b) uses the RPi Camera Web Interface software `https://elinux.org/RPi-Cam-Web-Interface`, which allows fast shutter speeds, concurrent camera access and automated exposure settings. Camera recordings are obtained in single image or continuous mode (max 90 frames/s) depending on user-defined settings.

The VHF analysis module (Figure 4.18c) uses the signal detection algorithm described by Gottwald et al. [Got+19]. When a new signal is received, its strength and duration are evaluated such that remote and noisy signals are filtered out. All other signals are classified as active

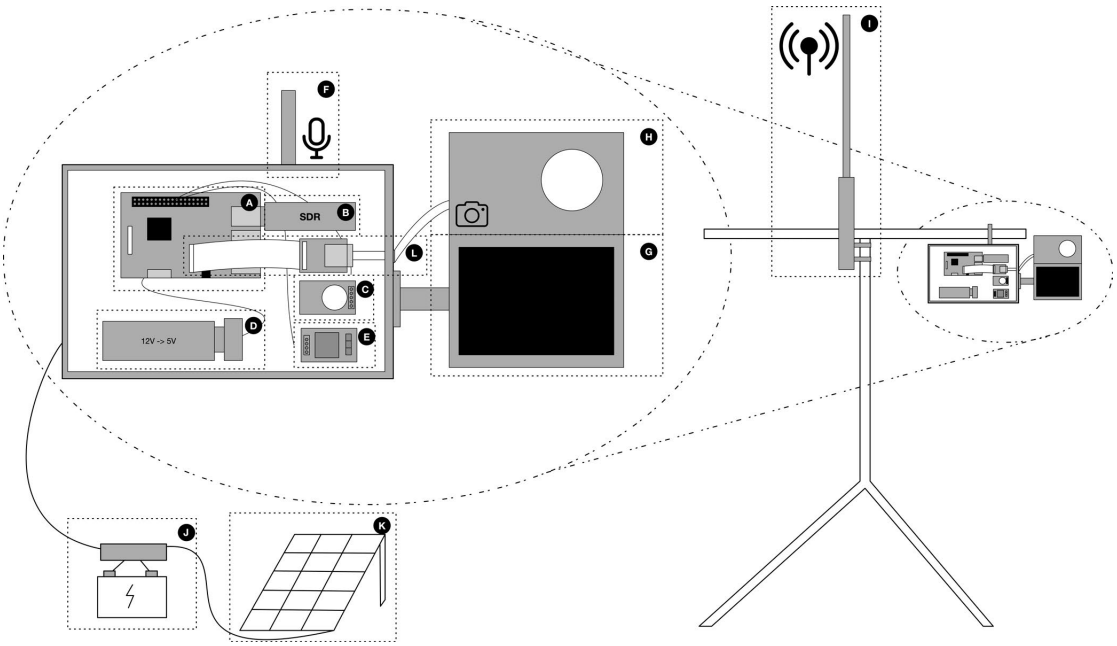


Figure 4.17: Hardware components of BatRack: (a) Raspberry pi mini computer, (b) rtl-sdr dongle, (c) real time clock or LTE stick (d) 12V to 5V converter with USB power supply, (e) KY-019 relay, (f) ultrasonic microphone, (g) IR spotlight, (h) Raspberry pi camera (NoIR or HQ-camera with removed IR filter), (i) omnidirectional antenna, (j) 12 V battery, (k) solar panel.

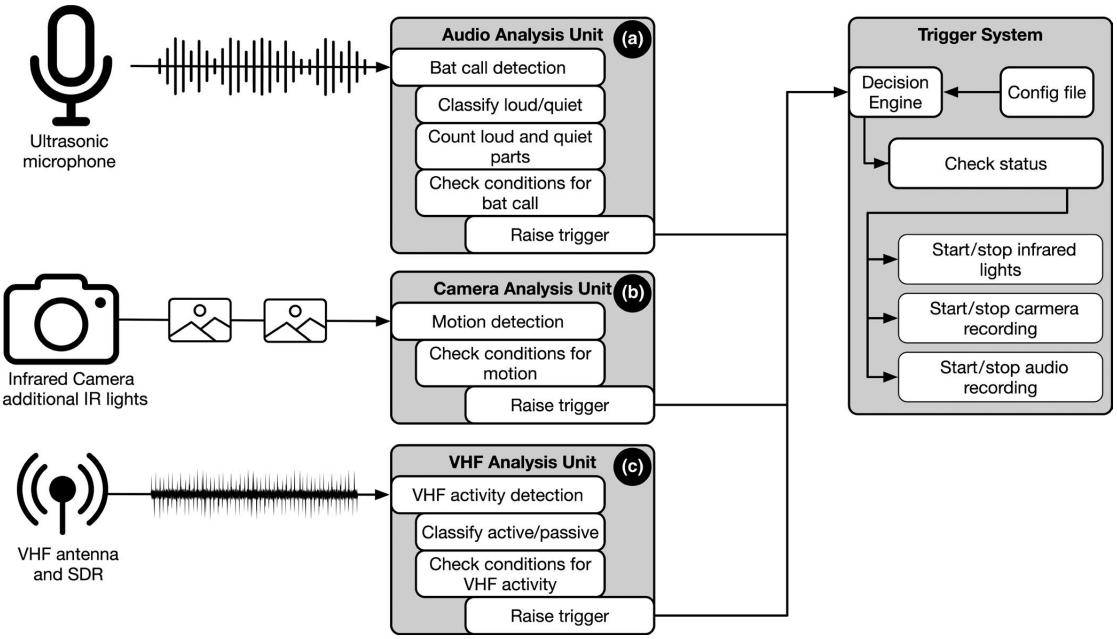


Figure 4.18: Analysis units of BatRack: (a) audio analysis unit (AAU), (b) camera analysis unit (CAU), (c) VHF analysis unit (VAU).

(i.e. flying) or inactive (i.e. resting; [Kay+11]). A bat is inactive if a standard deviation in signal strength <2 is detected over at least 30 s, and active otherwise (Figure 4.19). If the VHF signals are used to trigger audio or video recordings, only time periods with active signals are written to memory, thus saving storage space and reducing the number of recordings that must be analysed. The operational modes of the analysis modules can be scheduled and configured individually.

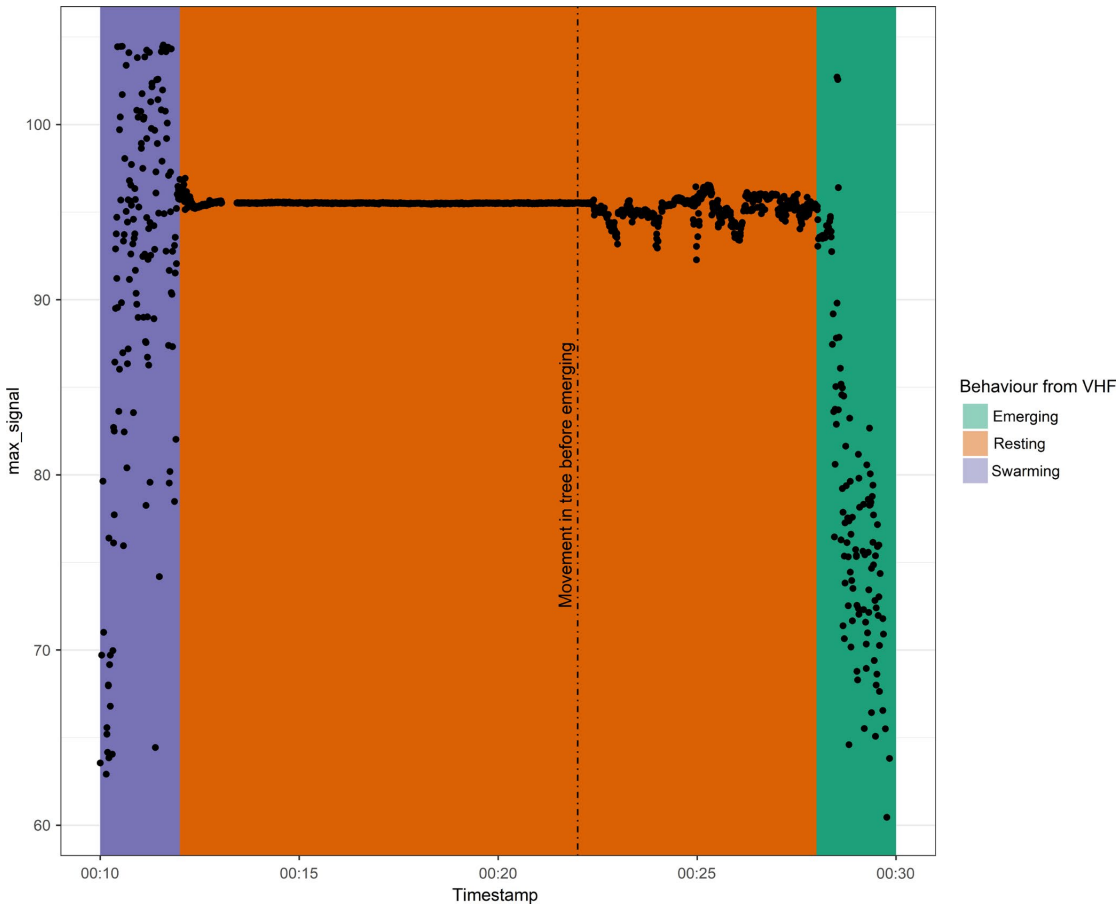


Figure 4.19: VHF signal patterns in relation to the different modes of behaviour. Swarming (purple), passive (orange), emerging from roost (green).

Audio-triggered video recordings

The suitability of passive ultrasonic audio observations for triggering video recordings of bats was tested by placing BatRack in front of a known roost of Bechstein's bats Myotis bechsteinii for one night. The use of highly sensitive settings (10 dB, 15 kHz) resulted in the triggering of video and audio recordings for 2 s approximately every 10 s. Audio recordings were visualized using BatScope [OB18] and classified as Bechstein's bats, other bats or no bats. Video sequences were manually screened for bats, and the ratio of simultaneous audio and video detections of bats served as the test variable.

To optimize the trigger parameters for the detection of Bechstein's bats, the recorded audio files were postprocessed using the trigger algorithm of the audio analysis unit. All possible combinations in the range of 15–50 kHz for the frequency threshold and 15–50 dB for the sound-pressure threshold were tested. All audio recordings classified as Bechstein's bat calls were treated as true positives; all other bat calls were excluded from the training dataset. The maximum F1 score, which is the harmonic mean of precision and recall, was used to select the parameter combination that best minimized false positives while correctly identifying most true positives.

VHF-triggered video recordings and individual-related behavioural patterns

To test the suitability of the VHF recordings in camera control and of the VHF signal strength in inferring behaviour, three pairs of Bechstein's bats from the same maternity colony were captured with mist nets between June and July 2020 and fitted with VHF tags. All females, except individual h172498, were in the expected reproductive state at the time of capture (Table 4.3). To monitor the bats, three BatRacks were placed in front of known roosting trees at a distance of 5–15 m for a total of 30 nights between June and August 2020.

ID	Reproductive state	Pair	Capture date	VHF frequency
h146480	Pregnant	Pair1	08.06.2020	150.187
h172494	Pregnant	Pair1	08.06.2020	150.128
h172498	Not reproducing	Pair2	23.06.2020	150.172
h146482	Lactating	Pair2	18.06.2020	150.199
h146486	Postlactating	Pair3	15.07.2020	150.199
h146488	Postlactating	Pair3	15.07.2020	150.156

Table 4.3: Studied female Bechstein's bat individuals and their pair assignments

To determine the suitability of VHF receptions in triggering video recordings of tagged individuals, the ratio of expected captures based on VHF patterns to manually screened, actual video captures of at least one visible bat was used as the test variable. To investigate the potential of individual measurements to infer behavioural patterns, in this case swarming and emerging, the VHF data were analysed manually. Swarming was defined as both a signal pattern indicating an active bat and a signal strength above a threshold of -20 dBW for at least 30 s (Figure 4.19, purple). This corresponded to the continuously flying of a tagged individual in close proximity to the sensor unit. Emerging was defined as a resting phase immediately followed by a strong signal, which in turn dropped off very quickly and did not stabilize immediately (Figure 4.19, green).

The observed behaviour of the bats was manually labelled as swarming if the recorded video sequences revealed an individual that moved back and forth in the area of the roost, if the individual briefly approached the tree, or if it left the roost after a short entry. Exits that were not directly followed by re-entry or swarming were classified as emerging.

To determine whether the video-captured bats could be identified as the tagged individuals, the VHF signal patterns and the corresponding movement patterns in the video were compared. Exemplary VHF data and video frames are shown in Figure 4.20. The full video sequence and animated VHF data are provided at the BatRack webpage.

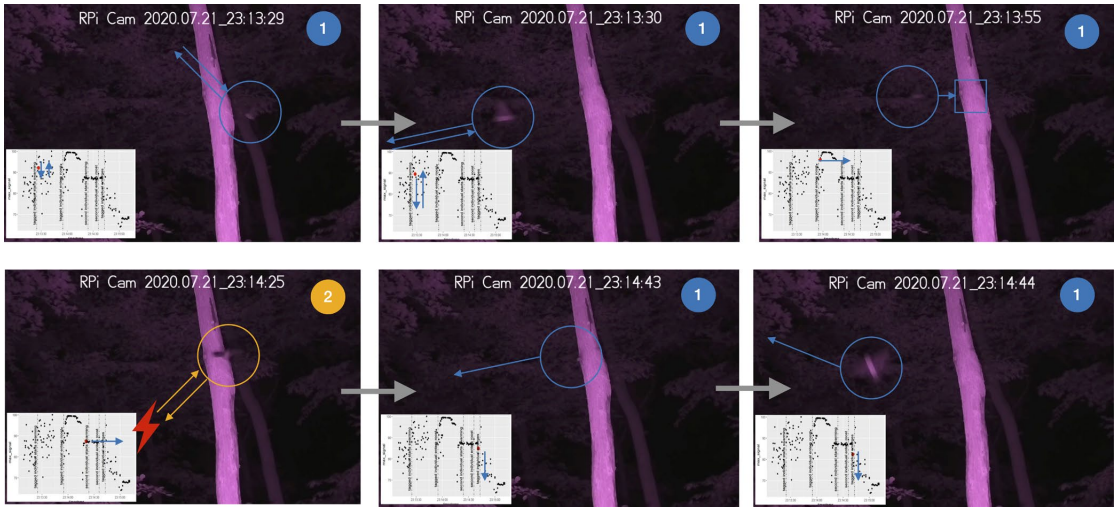


Figure 4.20: Identification of a tagged individual. The VHF signal shows strong fluctuations during swarming (up left and mid). The signal fluctuations decrease significantly after the bat enters the tree (up right, down left). Shortly after a second individual enters the tree (down mid), the tagged bat emerges from the tree (down right).

Unless otherwise stated, all analyses were performed using R [R F20].

4.3.4 Experimental Evaluation

Audio trigger performance

During the test night, 3,317 audio-triggered audio and video recordings with an average length of 2 s were collected. Bats could be manually identified on 170 video (5.1%) and 663 audio (20%) recordings. From the latter, 272 recordings (41%) most likely originated from Bechstein's bats. For 166 of the 170 (97.6%) videos, a bat call was recorded simultaneously; in 160 cases (94.1%), the call was classified as that of a Bechstein's bat.

After all files with calls that could not be assigned to Bechstein's bats were removed and files without bat calls retained as true negatives, the remaining 2,929 files were processed to determine the optimal trigger parameters. The optimal combination of frequency and volume threshold based on the maximum F1 score (0.91) was 15 dB and 38 kHz. With this combination, 235 out of 274 files containing Bechstein's bat calls (sensitivity = 0.858) were correctly identified; only 5 out of 2,655 negatives were falsely identified as positives (specificity = 0.998).

VHF trigger performance

In total, 205 video recordings were captured that matched the VHF sequences classified as swarming or emerging. Of these, 130 (63%) were considered to show swarming and 75 (37%) emerging. Manual screening of the footage revealed one or more bats on 170 of the 205 (83%) sequences. Swarming was successfully detected in 93% of the sequences in which swarming was expected (122 out of 130), and emerging in 65% of the sequences (49 out of 75).

From the 170 (91%) video detections of a bat, in 155 the bat could be identified as the tagged individual with a very high probability, based on comparison of the movement pattern with the VHF signal strength. Among the 49 emerging and 130 swarming events, this was the case for 47 (96%) and 108 (83%), respectively.

Individual behaviour patterns

Pregnant and postlactating bats (Figure 4.21; pairs one and three) did not show any apparent differences in their swarming and resting patterns, either between individuals of a pair or between pairs. All four individuals showed a higher swarming frequency on the night of the roost change and on the following night. During the latter, repeated swarming sequences and resting phases at the abandoned tree occurred. The lactating female (Figure 4.21; pair two, h146482), showed a higher frequency of swarming behaviour and resting periods than observed in the nonreproducing female (Figure 4.21; pair two).

4.3.5 Summary

A prerequisite for understanding dynamic natural environments as socio-ecological systems is data collected using highly automated monitoring systems. Recent developments have shown that the integration of (multiple) sensors and technologies in data acquisition and analysis can provide deep insights into the ecology of different species [GY19; Rip+20; Sch+19; Tol+20]. BatRack offers a highly promising solution in the observation of bats. With its modular COTS design, BatRack can be readily built and easily maintained, allows individual configurations and extensions, and enables both flexible scheduling and the combination of measurements.

BatRack yields reliable occurrence information based on audio or VHF recordings. The latter can be used in the retrieval of basic behavioural information even from a single, nondirectional VHF receiver. Especially for small bat species, the use of either VHF or audio to trigger a video unit results in more energy- and storage-efficient video capture than allowed by purely schedule-based recording. A high detection probability and a substantial reduction in false positives are ensured by applying targeted trigger parameters to the audio unit. Triggering based on the VHF signal results in an even better performance with bats captured almost all the time when one of the tagged bats triggered the recording. In our study, valuable video recordings were obtained even for bats flying up to 15 m away from the sensor.

Our case study on the behaviour patterns of Bechstein's bats illustrates the potential of BatRack. The observations provide first anecdotal indications of an association of increased swarming activity with a change of roost (pair one, three) and weening of the pup (pair two). However,

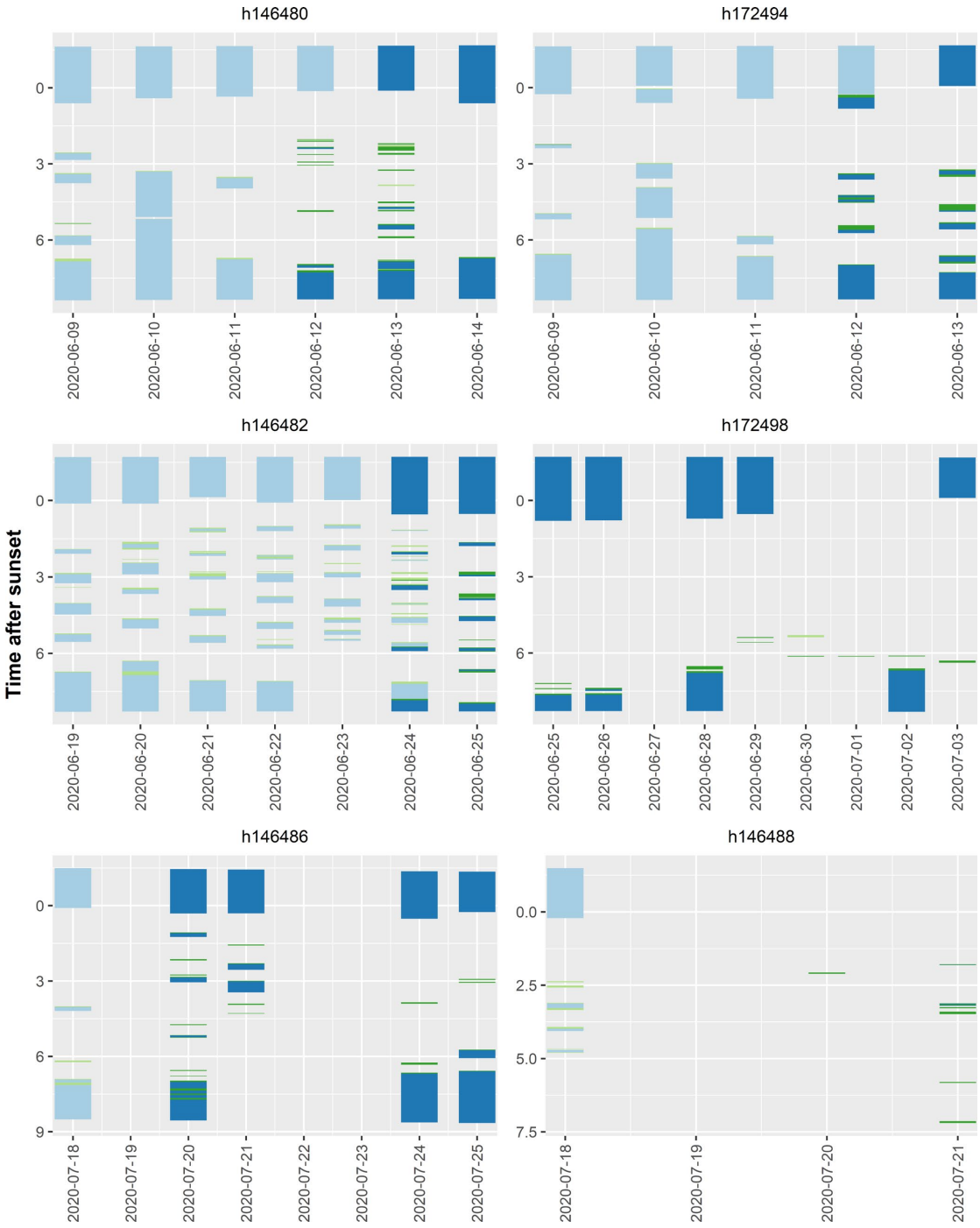


Figure 4.21: VHF-signal-derived behavioural patterns of Bechstein's bat pairs. Blue = inactivity, green = swarming. Gradations in the respective colour scale indicate the roost used for resting (Tree A = lighter blue, Tree B = darker blue) or for swarming (Tree A = lighter green, Tree B = darker green).

the advantage of information acquired at the individual level comes at the price of having to

tag the animals. Furthermore, the identification of an individual is more difficult if several tagged individuals with similar levels of activity are recorded simultaneously.

The application possibilities of BatRack are manifold. Observations that were previously only possible in the laboratory can be obtained in natural habitats. BatRack is best suited for studies where the observation of bats is linked to a specific and small area (e.g. hibernation roosts, maternity colonies, specific resource occurrences). The focus here is on the study of social behaviour between animals or the use of resources. The mobility of BatRack also makes it possible to complement laboratory studies with field experiments (e.g. changes in resource availability). Moreover, while behavioural contexts can often be inferred from vocalizations, reference recordings are missing for many species [TMR19]. This deficiency can be addressed by BatRack, which can be used to collect visual ground truth of the behavioural significance of vocalizations. Thus, BatRack is a promising building block to close knowledge gaps regarding bat behaviour and to develop and evaluate conservation measures.

With regard to the classification in Figure 4.1 on page 26, BatRack cannot be classified in a straightforward manner. The intersection of the different sensors enables a functionality that was hardly possible before, even with manual techniques. Manual VHF telemetry, for example, can be used to determine the roosting locations of bats, but immediate triggering of a camera trap is not possible due to the high delay. The recording of bat calls with the help of audio recorders is also technically possible, but the possibility of triggering a camera trap based on this is also missing here. Finally, there is the camera trap itself, which can be triggered in response to movement, but which generates many false positives. The classification in the Figure 4.1 refers to the enhancements of the video recordings themselves, which previously occurred only in small numbers or with high false positive rates.

4.4 *Bird@Edge*: Bird Species Recognition at the Edge

4.4.1 Introduction

The continuous loss of biodiversity is particularly evident from the sharp decline of bird populations in recent decades. Birds are important for many ecosystems, since they interconnect habitats, resources, and biological processes, and thus serve as important early warning bioindicators of an ecosystem's health. Thus, changes in bird species in time and space should be detected as early as possible.

Traditionally, this is achieved by human experts who walk around a natural habitat to look at birds and listen to bird sounds, identify bird species present in the sounds, and take notes of their occurrence. In recent years, this is often supported by placing microphones in the natural habitats of birds and recording their sounds. The audio data recorded in this way is then evaluated either manually by human experts or by means of automatic analysis methods to recognize bird species in soundscapes. The disadvantages of this approach are: (a) there is a potentially large amount of recorded audio data that can usually only be evaluated after the end of the recording time, and (b) there is an inherent time delay between recording the audio data and delivering the recognition results.

In this section, we combine Edge Computing and Artificial Intelligence (AI) to present Bird@Edge, an *Edge AI* system for recognizing bird species in audio recordings to support real-time biodiversity monitoring. Bird@Edge is based on embedded edge devices operating in a distributed system to enable efficient, continuous evaluation of soundscapes recorded in forests. Multiple microphones based on ESP32 microcontroller units (called Bird@Edge Mics) stream audio to a local Bird@Edge Station, on which bird species recognition is performed. The recognition results of different Bird@Edge Stations are transmitted to a backend cloud for further analysis, e.g., by biodiversity researchers.

To recognize bird species in soundscapes, a deep neural network based on the EfficientNet-B3 architecture is trained and optimized for execution on embedded edge devices and deployed on a NVIDIA Jetson Nano board using the DeepStream SDK. Our experimental results show that our deep neural network model outperforms the state-of-the-art BirdNET neural network on several data sets and achieves a recognition quality of up to 95.2% mean average precision on soundscape recordings in the Marburg Open Forest, a research and teaching forest of the University of Marburg, Germany. Measurements of the power consumption of a Bird@Edge Station and the Bird@Edge Mics highlight the real-world applicability of the approach. All software and firmware components of Bird@Edge are available under open source licenses[24]. Our contributions are:

- We present a novel Edge AI approach for recognizing bird species in audio recordings; it supports efficient live data transmission and provides high-quality recognition results.
- We propose a deep neural network based on the EfficientNet-B3 architecture optimized for execution on embedded edge devices to identify bird species in soundscapes.
- We evaluate our Edge AI approach in terms of recognition quality, runtime performance, and power consumption.

Parts of this section have been published in Jonas Höchst, Hicham Bellafkir, Patrick Lampe, Markus Vogelbacher, Markus Mühling, Daniel Schneider, Kim Lindner, Sascha Rösner, Dana G. Schabo, Nina Farwig, and Bernd Freisleben. "Bird@Edge: Bird Species Recognition at the Edge." in: *International Conference on Networked Systems (NETYS)*. Springer. May 2022. DOI: 10.1007/978-3-031-17436-0_6.

4.4.2 Related Work

In this section, we discuss related work with respect to current machine learning approaches for bird species recognition in audio recordings and edge AI approaches for biodiversity monitoring.

Bird Species Recognition

For many years, bird populations were monitored manually by ornithologists who identified birds visually and acoustically on site. The introduction of autonomous recording units (ARU)

[24] https://github.com/umr-ds/BirdEdge

opened new possibilities. Although such passively recorded data does not provide any visual information, the resulting bird surveys conducted by humans from sound recordings are comparable to traditional monitoring approaches in the field [Dar+18].

Furthermore, machine learning methods, such as Convolutional Neural Networks (CNN), are increasingly being used for automatically recognizing bird species in soundscapes. For example, BirdNET is a task-specific CNN architecture trained on a large audio data set using extensive data pre-processing, augmentation, and mixup that achieves state-of-the-art performance [Kah+21b]. The audio spectrograms are generated using a Fast Fourier Transform (FFT) with a high temporal resolution. BirdNet is based on a ResNet [He+16] architecture and is capable of identifying 984 North American and European bird species.

More recently, BirdNET-Lite[25] has been released. This neural network is optimized for mobile and edge devices and can recognize more than 6,000 bird species. It takes raw audio as its input and generates spectrograms on-the-fly. Mühling et al. [Müh+20] proposed a task-specific neural network created by neural architecture search [ZL17]. It won the BirdCLEF 2020 challenge [Kah+20]. It also operates on raw audio data and contains multiple auxiliary heads and recurrent layers.

Recently, Vision Transformers (ViT) achieved great improvements in computer vision tasks [Dos+21] and audio event classification[GCG21]. Puget [Pug21] adopted a ViT architecture for bird song recognition and achieved results comparable to CNNs. However, the annual birdcall identification challenge (BirdCLEF [Kah+21a]) is currently dominated by approaches based on CNNs. The top approaches typically use ensembles of CNNs and heavy parameter tuning. The winning approach at BirdCLEF 2021, for example, uses Mel spectrograms, network architectures based on ResNet-50 [He+16], and gradient boosting to refine the results using metadata. The runners-up Henkel et al. [HPS21] presented an ensemble of nine CNNs. During training, they used 30 second Mel spectrograms to mitigate the effect of the weakly labeled training data and applied a novel mixup scheme within and across training samples for extensive data augmentation. Furthermore, a binary bird call/no bird call classifier contributed to the final result. However, combining several machine learning models leads to a considerably increased computational effort.

Edge AI for Biodiversity Monitoring

Executing machine learning algorithms on edge devices leads to a quantitative increase of data through continuous observation, where previously only individual data points could be collected with manual effort, often including a bias of individual experiences depending on, e.g., habitat or bird species. Merenda et al. [MPI20] survey several approaches based on the execution of machine learning methods on hardware with limited resources. Gallacher et al. [Gal+21] deployed 15 sensors in a large urban park to process recorded audio data of bats locally, which allowed monitoring their activities for several months. Given that the system has only been operated in an urban environment, the limitations of this approach are that network connectivity must be available via WiFi, and that a fixed power supply must be present. Novel deep learning approaches presented by Disabato et al. [Dis+21] further improved

[25] `https://github.com/kahst/BirdNET-Lite`

bird song recognition at the edge. These approaches provide high accuracy while reducing computational and memory requirements, with limited battery lifetimes of up to 12.4 days on an STM32H743ZI microcontroller. Likewise, Zualkernan et al. [Zua+21] compare different edge computing platforms based on neural networks using bat species classification as an example. While the NVIDIA Jetson Nano is the only device capable of executing a TensorRT model on its GPU, both the Raspberry Pi 3B+ and the Google Coral showed good results when executing a reduced TensorFlow-Lite model.

4.4.3 Bird@Edge

Bird@Edge is designed as an *Edge AI* system based on distributed embedded edge devices to enable efficient, continuous evaluation of soundscapes recorded in forests. Multiple Bird@Edge Mics stream audio wirelessly to a local Bird@Edge Station, on which bird species recognition is performed. The recognition results of different Bird@Edge Stations are transmitted to a backend for further analysis. The results are stored in a time series database and can be visualized, as shown in Fig. 4.22. Using hidden microphones also supports recognizing very elusive species that are hard to detect while ecologists are present in field to conduct a census.

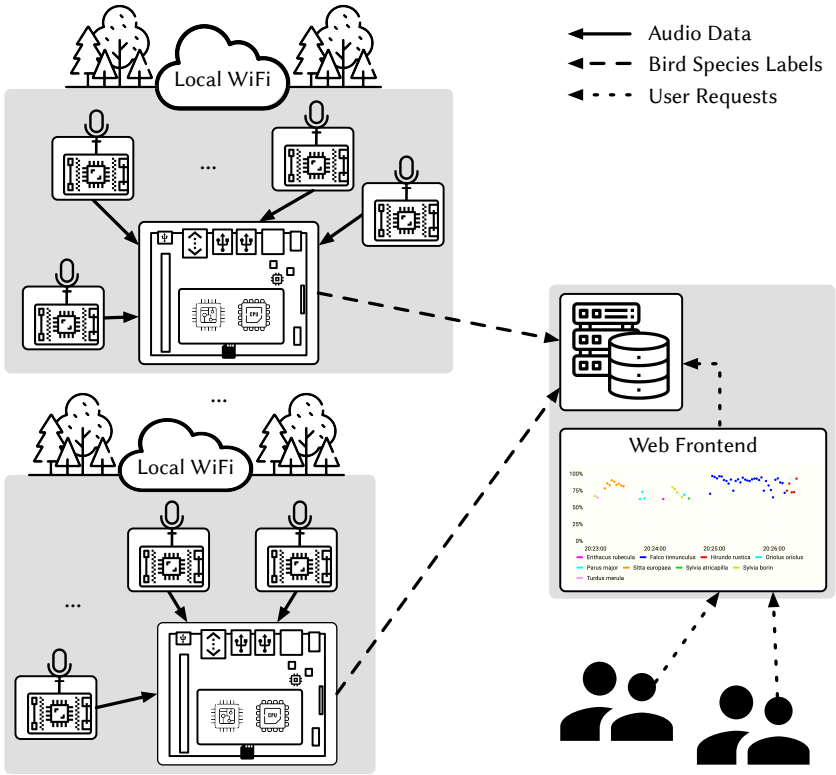


Figure 4.22: Overview over the Bird@Edge system

A Bird@Edge Station consumes significantly more power than a microphone node, but can run a neural network for bird species recognition for more than one audio stream. We can feed 1 to 10 audio streams into the neural network and thus operate a variable number of

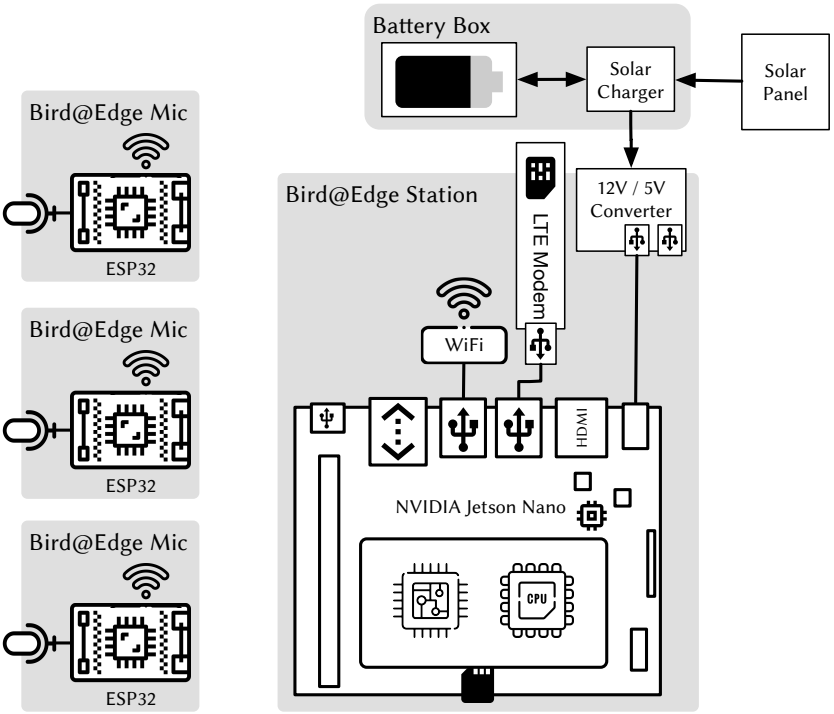


Figure 4.23: Bird@Edge hardware components

Bird@Edge Mics at one Bird@Edge Station. Different numbers of Bird@Edge Mics may be present when a new microphone node appears (e.g., by switching it on) or leaves (e.g., due to battery shortage).

To generate a list of bird species at a Bird@Edge Station, chunks of an incoming audio stream are passed to the neural network, which may return multiple results, since we process mixtures of recorded bird songs, i.e., soundscapes. These potentially multiple results per audio segment are then collected and aggregated into larger intervals in the time series database in the backend cloud. The size of the interval can be dynamically changed and visualized in near real-time. In addition, the status of the microphone nodes and potential problems can be detected much faster than collecting data only every few days.

Bird@Edge Hardware

The hardware used for Bird@Edge consists of (a) Bird@Edge Mics, which are in charge of recording and transmitting audio at the deployed location; (b) Bird@Edge Stations, which receive audio streams from multiple Bird@Edge Mics and execute the Bird@Edge processing pipeline. Figure 4.23 provides an overview of the hardware components used in Bird@Edge.

A Bird@Edge Mic consists of an Espressif ESP32 microcontroller that has a dual-core CPU running at 80 MHz, Bluetooth and WiFi connectivity, as well as multiple input and output options, including an I2S (Inter-IC Sound) bus. Connected to it is a Knowles SPH0645LM4H

microphone capable of recording audio in the range between 50 Hz and 15 kHz[26]. A Bird@Edge Mic can be powered either using single 18650 Li-ion cells or using one of the widely available USB power banks. The price of a Bird@Edge mic of 22€ to 50€ is composed of the ESP32, depending on the offer and model 5€ to 15€, the I2S microphone 7 - 12€ and a battery for 10€ - 20€. All components can be placed in a small case of 10 x 10 x 5 centimeters, which does not exceed the weight of 500 grams.

At the heart of a Bird@Edge Station is a NVIDIA Jetson Nano. It allows the efficient execution of machine learning models in a low power environment. A Realtek RTL8812BU-based USB WiFi is used to enable wireless networking with the board and allow connection to the Bird@Edge Mics. In addition, a Huawei E3372H LTE modem is installed to connect to the Internet in rural areas. The station is powered by 12V solar battery system connected to the Jetson Board via a 12V/5V step down converter. The hardware of a Bird@Edge Station costs about 110€, with 50€ for the Jetson Nano, 20€ for the USB WiFi adapter, and 40€ for the LTE modem. The components of an Bird@Edge Station, including a solar charge controller, can be fitted into an industrial enclosure measuring 25 x 18 x 12 centimeters, weighing less than 1.5 kilograms in total.

Bird@Edge Software

Bird@Edge consists of a variety of software components that enable its smooth configuration and operation. Figure 4.24 shows these software components, as well as the data flows and interaction possibilities of the users with the system.

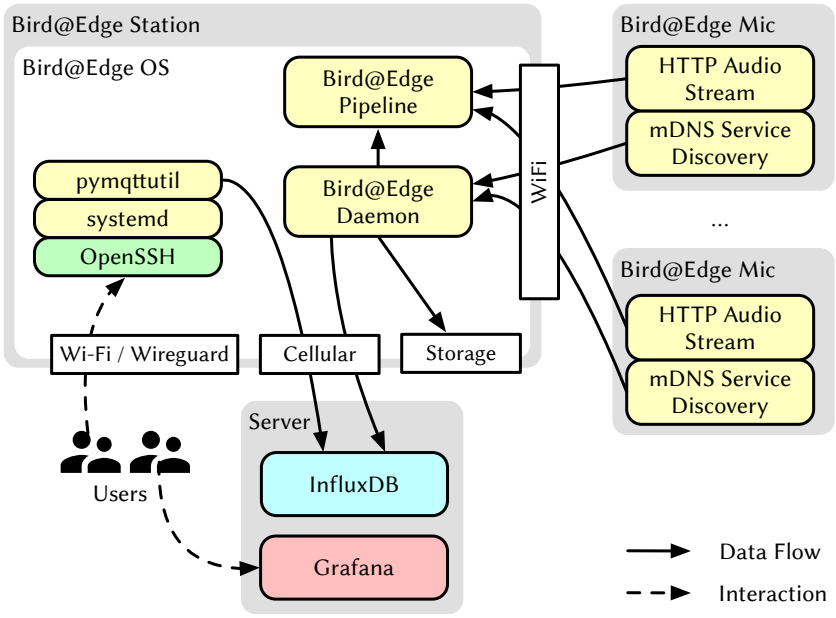


Figure 4.24: Bird@Edge software components

[26] https://www.knowles.com/docs/default-source/default-document-library/sph0645lm4h-1-datasheet.pdf

The software for the Bird@Edge Mics is built using components of the Espressif Development Framework (ESP-IDF), i.e., HTTP Server, Multicast DNS Implementation, and I2S drivers. When booting up, the Bird@Edge Mic connects to the WiFi network (SSID: BirdEdge) with the best signal strength and reports its own accessibility via the service identifier mDNS. Then, the HTTP server is started, which provides the audio stream of the microphone for the Bird@Edge station. To detect connection interruptions, the WiFi connection is also checked at intervals of one second with the aid of ICMP and, if necessary, the WiFi connection is re-established. The Bird@Edge Mic software is available online[27].

The software running on a Bird@Edge Station is based on the NVIDIA Jetson Nano Development Kit Operating System, which in turn is based on Ubuntu Linux. The central component responsible for detecting the Bird@Edge Mics, executing the processing pipeline and transmitting the results is called birdedged (Bird@Edge Daemon). It continuously searches for newly connected Bird@Edge devices and restarts the processing pipeline accordingly when devices are found or dropped. Bird species recognition results from the processing pipeline are captured and transmitted to the InfluxDB server system. The server system that collects data from potentially multiple Bird@Edge implementations runs Grafana, a dashboard visualization WebUI designed specifically for stream data[28].

The operating system running on Bird@Edge Stations is built using PIMOD [Höc+20b] and is available online[29]. NVIDIA's licenses do not allow to redistribute complete operating system images, however pimod allows to reduce the necessary steps and easily create the images.

4.4.4 Recognizing Bird Species in Soundscapes

In this section, we describe our deep learning approach to bird species recognition in audio recordings including the preprocessing steps, the neural network as well as its optimization and deployment on the NVIDIA Jetson Nano edge device. The deep neural network model is designed to recognize 82 bird species indigenous in Germany and background noise that is typical for German forests.

Audio Preprocessing

We selected 44.1 kHz as the sampling rate and analyzed frequencies up to 22.05 kHz to cover the frequency ranges of the bird song patterns. The task is considered as a classification problem, aiming to recognize bird species in 5-second audio snippets. To avoid overfitting and enrich our data set, we randomly select these 5-second snippets and add randomly selected noise from up to four background samples. This encourages our model to focus on the patterns that are important for species recognition. The recognition is based on visual representations of the frequency spectrum as it changes over time, called spectrograms. In our case, we use Mel spectrograms that are generated using 128 Mel bands and an FFT window size of 1,024.

[27] `https://github.com/umr-ds/BirdEdge/tree/main/BirdEdge-Client`
[28] `https://grafana.com`
[29] `https://github.com/umr-ds/BirdAtEdge-OS`

Neural Network Architecture

Our approach to recognize bird species relies on an EfficientNet-B3 [TL19] architecture pre-trained on ImageNet [Rus+15]. The model is fine-tuned in two phases to target domain using the Adam [KB15] optimizer. In the first phase, we only train the last, randomly initialized layer for 40 epochs with an initial learning rate of 0.004, while the remaining layers with pre-trained weights are frozen. In the second phase, we train all layers of the model until convergence, while reducing the initial learning rate by a factor of 10. Furthermore, a binary cross-entropy loss combined with modulation factors motivated by the success of focal loss [Lin+17] in the field of object detection are used to emphasize difficult samples during the training process. Since the underlying data set is only weakly labeled, we use positive training samples for one species as negative samples for the others. Furthermore, samples labeled negative from expert feedback are defined as hard negatives in the following. Our loss function is defined as follows:

$$L = \sum_{k=1}^{K} l(y_k, p_k),$$

$$l(y, p) = \begin{cases} -\alpha_{pos}(1-p)^{\gamma}\log(p) & \text{if } y \text{ is positive} \\ -\alpha_{n}p^{\gamma}\log(1-p) & \text{if } y \text{ is negative} \\ -\alpha_{hn}p^{\gamma}\log(1-p) & \text{if } y \text{ is hard negative} \end{cases}$$

where K is the number of bird classes, p_k is the predicted probability for the k-th class, y_k is the k-th ground truth label, α_{pos} is the weighting factor for positive labels, α_n for negative or undefined labels, α_{hn} for hard negative labels and γ is the focusing parameter.

We implemented our approach using the TensorFlow deep learning framework [Mar+15]. For audio processing and especially spectrogram generation, we use the librosa library [McF+15].

Optimizing the Neural Network for Edge Devices

To speed up inference, we optimized our model using the TensorRT[30] library. This library includes an inference optimizer for CUDA-capable target devices that applies various operations, such as quantization and memory optimization, to reduce the inference time. In particular, the floating point precision is reduced by quantizing to FP16 or INT8, while maintaining high accuracy. We optimized our model by using FP16 quantization in addition to the original FP32 weights, since the NVIDIA Jetson Nano does not support INT8 computations natively. Furthermore, we applied target-specific auto-tuning to select the best algorithms and quantization for each layer.

[30] `https://developer.nvidia.com/tensorrt`

Inference

We use the DeepStream SDK[31] to deploy our optimized model on the NVIDIA Jetson Nano board with high throughput rates. DeepStream is based on the GStreamer framework and provides a pipeline that takes an input stream and performs hardware accelerated inference on it. An overview of our pipeline composed with DeepStream is presented in Figure 4.25. First, the

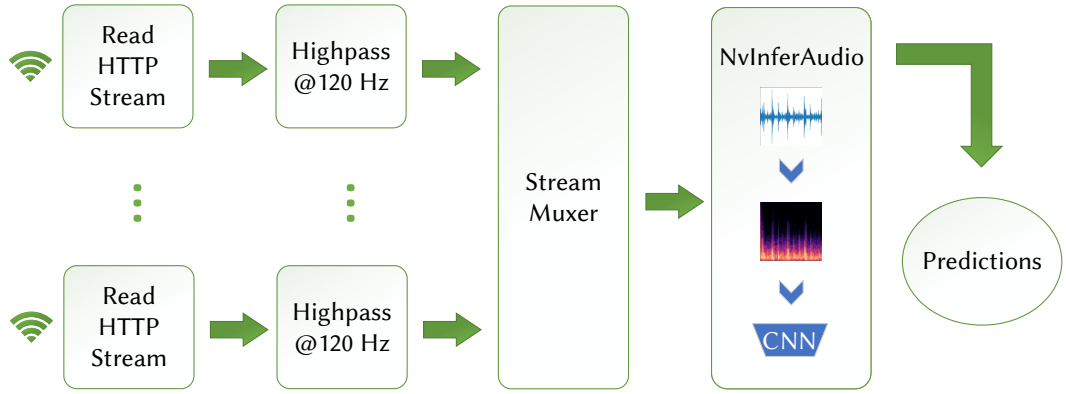


Figure 4.25: Overview of the Bird@Edge processing pipeline

N HTTP streams are read and parsed from the WiFi signal. Since the microphone we used (see Section 4.4.3 for details) induces noise in the lowest frequency bands, we apply a highpass filter that attenuates all frequencies below 120 Hz to each stream. These frequencies are irrelevant for bird species recognition and can therefore be neglected. We prefer the Chebyshev highpass filter over the windowed sinc filter, because it is much faster. Next, we use DeepStream's stream muxer to bundle our streams into one batch and forward the data to the NvInferAudio plugin. This plugin provides inference for audio streams and automatically generates the respective Mel spectrograms. Finally, the spectrograms are passed to our model with a batch size of N, and the obtained predictions are retrieved. To be able to process the streams in real-time with a high temporal resolution, we take 5 second snippets with a stride of one second.

4.4.5 Experimental Evaluation

In this section, we present experimental results in terms of (a) bird species recognition quality and execution speed, (b) visualization of bird recognition results, as well as (c) power consumption measurements of a Bird@Edge Station and a Bird@Edge Mic.

[31] `https://developer.nvidia.com/deepstream-sdk`

Bird Species Recognition Quality and Execution Speed

Data Sets.

Our neural network models were evaluated and compared to BirdNET [Kah+21b] and BirdNET-Lite[25] on data sets collected from three sources. We recorded a first data set with AudioMoth devices [Hil+19] in the Marburg Open Forest (MOF). The recordings were labeled on a 5 second basis by human experts. In total, 33 species occur in the MOF data set. Since the amount of labeled data in the MOF data set is not sufficient to train a deep learning model, we acquired further data sets by crawling data from the online bird song collections Xeno-Canto [Xen] and iNaturalist [iNa]. The assets included in these data sets have often higher quality and contain less background noise. In our evaluation, we took up to 10% of the files of each class. To make sure that we do not feed snippets without bird calls, we first applied the heuristic used by Kahl et al. [Kah+21b] and selected up to three 5 second snippets containing a bird call for each test file. Table 4.4 shows an overview of the training and test data.

Data Set	MOF	Xeno-Canto	iNaturalist
Training	4,294	104,989	30,631
Test	913	2,144	1,365

Table 4.4: Overview of the training and test data

Quality Metrics.

To evaluate the performance of our bird species recognition approach, we use average precision (AP) as our quality metric. The AP score is the most commonly used quality measure for retrieval results and approximates the area under the recall-precision curve. The task of bird call recognition can be considered as a retrieval problem for each species where the annotated audio samples represent the relevant documents. Then, the AP score is calculated from the list of ranked documents as follows:

$$AP(\rho) = \frac{1}{|R \cap \rho^N|} \sum_{k=1}^{N} \frac{\left|R \cap \rho^k\right|}{k} \psi(i_k),$$

$$\text{with} \quad \psi(i_k) = \begin{cases} 1 & \text{if } i_k \in R \\ 0 & \text{otherwise} \end{cases}$$

where N is the length of the ranked document list (total number of analyzed audio snippets), $\rho^k = \{i_1, i_2, \ldots, i_k\}$ is the ranked document list up to rank k, R is the set of relevant documents (audio snippets containing a bird call), $\left|R \cap \rho^k\right|$ is the number of relevant documents in the top-k of ρ and $\psi(i_k)$ is the relevance function. Generally speaking, AP is the average of the precision values at each relevant document. To evaluate the overall performance, the mean AP score is calculated by taking the mean value of the AP scores from each species.

Method	MOF	XC	iNat
BirdNET[Kah+21b]	0.833	0.725	0.725
BirdNET-Lite[25]	0.859	0.737	0.714
EfficientNet-B3	0.952	0.820	0.811
Bird@Edge	0.952	0.816	0.819

Table 4.5: Results (mAP)

Model	Device	Inference time (ms)
BirdNET-Lite[25]	Raspberry Pi-4B	279
Bird@Edge (FP32)	Jetson Nano	64
Bird@Edge	Jetson Nano	54

Table 4.6: Model inference runtimes

Results.

First, we evaluated the recognition quality of our models, namely the original trained model (EfficientNet-B3) as well as the optimized model (Bird@Edge) and compare the results to BirdNET and BirdNET-Lite. While EfficientNet-B3 is evaluated with TensorFlow on a workstation, the Bird@Edge model is run on the NVIDIA Jetson Nano for inference.

BirdNET and BirdNET-Lite take the recording location as additional metadata along with the corresponding audio input. As longitude and latitude, we take the coordinates of the Marburg Open Forest for all data sets, since we only use bird species resident in this specific forest for evaluation. Since the length of the audio input of the BirdNET models differs from our approach, the 5 second samples are split into two 3 second snippets with an overlap of 1 second and the results are averaged for the final prediction.

Table 4.5 summarizes the experimental bird species recognition results. Our original model (EfficientNet-B3) outperforms BirdNET-Lite as well as BirdNET by roughly 10% in terms of mAP on all data sets considered. While keeping the recognition quality, the optimized Bird@Edge model achieves an inference runtime of 64 ms per spectrogram, as shown in Table 4.6. Adding model quantization with 16-bit floating point precision where appropriate effectively reduces the inference runtime on the NVIDIA Jetson Nano board by 10 ms. We also compared the runtimes of our models to BirdNET-Lite. Similar to BirdNET-Pi[32], we ran BirdNET-Lite on a Raspberry Pi-4B with 4 CPU threads in parallel. Table 4.6 reveals that our setting is more than four times faster.

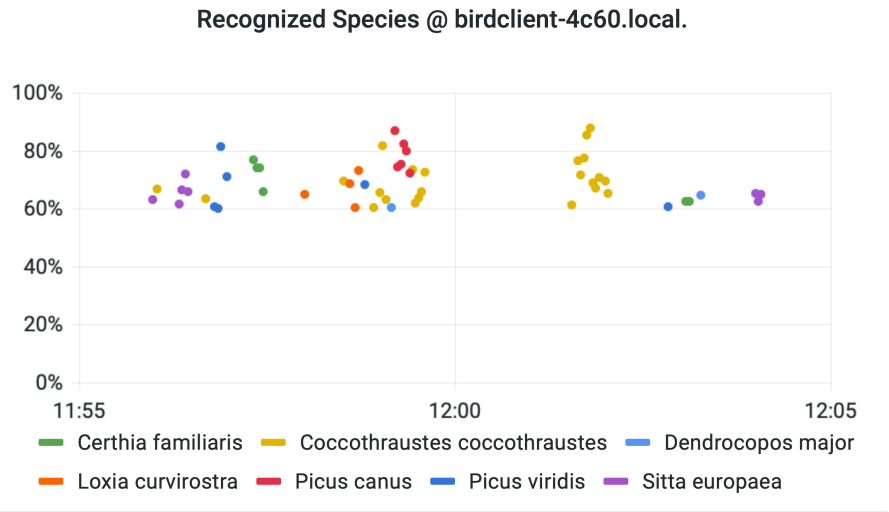


Figure 4.26: Grafana panel (x-axis: clock time; y-axis: recognition confidence) showing recognized bird species of a certain Bird@Edge Mic, based on Xeno-Canto file XC706150, recorded by user *brickegickel*

Visualization of Bird Species Recognition Results

Figure 4.26 shows a Grafana screenshot of an automatically generated graph of the recognized bird species of a Bird@Edge station. To generate the figure, the publicly available soundscape audio file XC706150 from Xeno-Canto[33] of the target area was played back and captured by the Bird@Edge Mic. The clock time is shown on the x-axis. The confidence of the recognition is plotted on the y-axis. The data is grouped according to the recognized bird species labels, distinguished by color. For every Bird@Edge Mic, a separate figure is generated, and its parameters, e.g., plotted time frame or selection of species, can be configured.

Some observations can be derived from this simple visualization. First, there are several recognized occurrences of *Coccothraustes coccothraustes* (hawfinch) in two clusters. *Picus canus* (grey-headed woodpecker) is detected multiple times over the duration of 12 seconds, and *Sitta europaea* (Eurasian nuthatch) is detected in two clusters each at the beginning and end of the observation period. All three observations indicate that individuals were heard on the recordings and were in the area at these times. For *Loxia curvirostra* (red crossbill) and *Dendrocopos major* (great spotted woodpecker), only 4 and 2 observations were made, respectively; these were probably heard only in the background. More sophisticated analyses can be performed based on researchers' requirements, such as heat maps of the occurrence of species based on their geo-positions, or time-based plots. This can include both short-term considerations, such as the time of day at which certain species are active, or long-term aspects, such as during which period a particular species is particularly active.

[32] `https://github.com/mcguirepr89/BirdNET-Pi`
[33] `https://xeno-canto.org/706150`

Power Consumption

An important aspect for the applicability of Bird@Edge in real applications is its power consumption. Therefore, we measured the power consumption of a Bird@Edge Station and a Bird@Edge Mic.

To measure the power consumption of a Bird@Edge Station, we used the internal power monitors of the NVIDIA Jetson Nano, since these enable the differentiation between CPU, GPU, and total power consumption. The power measurements were performed in different profiles: a) the 10 Watt maximum performance profile (default), b) the 5 Watt low power profile from NVIDIA, and c) a custom low power profile created for Bird@Edge. In this custom power profile, only 2 of the 4 CPU cores were used, running at a maximum frequency of 614 MHz, and the GPU was limited to a maximum of 230 MHz. As a baseline, the power consumption is measured with 5 connected Bird@Edge mics, and only the measured values while the pipeline is running are averaged. In this setup, the maximum performance mode requires 4.86 W, the low power profile 4.19 W and the custom low power mode only requires 3.16 W, i.e., roughly 35% less compared to the maximum performance mode with no performance degradation observable. Our observations during the execution of the experiments suggest that the GPU's dynamic frequency scaling algorithm tends to be too conservative to permanently lower the clock and thus prevents the possible lower power consumption.

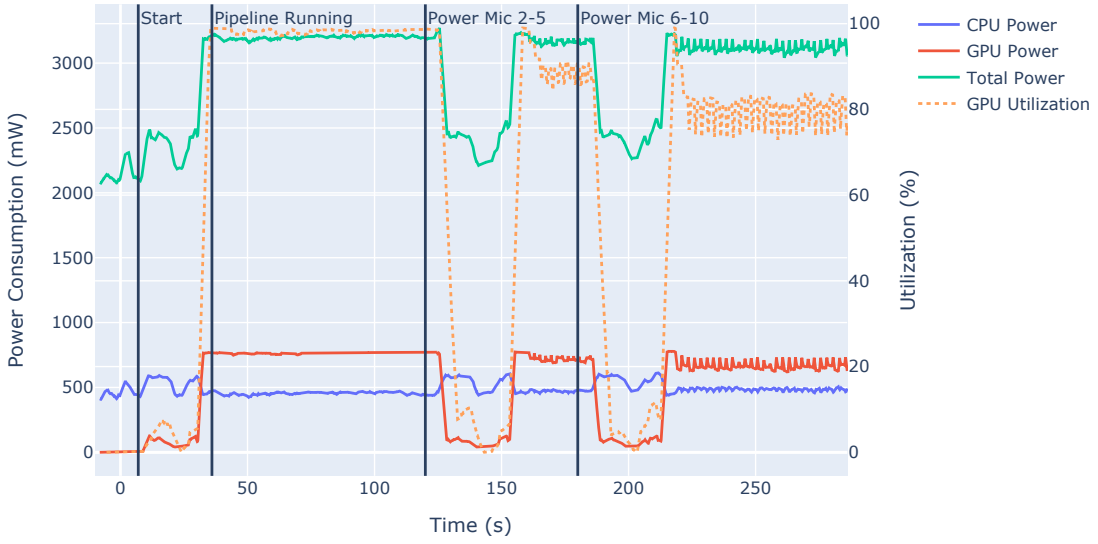


Figure 4.27: Power consumption of a Bird@Edge Station in a dynamic scenario.

Figure 4.27 shows the power consumption of a Bird@Edge station in a short scenario with a changing number of connected Bird@Edge Mics. At the beginning, the system is switched on, but the neural network for bird species is not running; the system needs 2.1 W in this state. At t=0, the neural network is started with a Bird@Edge Mic already connected to the station. The neural network model is loaded into memory from t=6, for which the CPU requires up to 0.59 W. From time t=36, i.e., 30 seconds after the start of the pipeline, the neural network model runs and forwards results to the backend. In this phase, the Bird@Edge station requires an average of 3.18 W. At t=120 and t=180, 4 and 5 additional Bird@Edge Mics are switched on, which first

connect to the Bird@Edge Station via WiFi, then are discovered via mDNS, which results in the reconfiguration of the processing pipeline and its restart. In both cases, the reboot took about 35 seconds, with 5 seconds for WiFi connection and discovery, and 30 seconds for pipeline reboot. With 5 and 10 Bird@Edge Mics connected, the Bird@Edge Station requires 3.16 W and 3.13 W, respectively. Particularly noteworthy is the station's lower power consumption when a larger number of Bird@Edge Mics are connected. Figure 4.27 indicates that GPU utilization is lower with many Bird@Edge Mics connected, compared to smaller numbers of Bird@Edge Mics (98% with 1 client, 88% with 5 clients, 80% with 10 clients). This is probably due to internals of the DeepStream SDK and may be influenced by the implementation, e.g., with respect to the handling of unused streams.

The magnitude of the Bird@Edge Station's power consumption necessitates the use of an LFP, VRLA or AGM battery, which at 12 Volts typically have a capacity of 50 to 200 Ah. The available 60 to 2400 Wh allow the operation of a Bird@Edge Station from 7 up to 31 days. In combination with a solar panel between 50 and 100 watt, a continuous operation is also possible during periods of weak sunlight.

To measure the power consumption of a Bird@Edge Mic, we used a Monsoon High Voltage Power Monitor[34] and connected the ESP32 using the 3.3 volt input. The measurements of a Bird@Edge Mic show a power usage of 665 mW whenever the stream is activated and data is sent to the station. The power measurements were performed with three different off-the-shelf ESP32 boards, since the additional electronics present on the boards can have an additional impact on power consumption. The three boards differed only slightly in terms of power consumption: 0.452 W was needed by the *Adafruit HUZZAH32*, 0.452 W by the *Joy-IT NodeMCU ESP32*, and 0.421 W by the *SparkFun ESP32 Thing Plus*. The latter[35] is the board of choice for our application, due to the lowest power consumption, a direct LiPo battery connector, and an external WiFi antenna.

To get a realistic estimation of the battery life of a Bird@Edge Mic, further measurements were performed with 3.7 Volt via the corresponding connectors for LiPo batteries. The SparkFun board required 0.468 W or 126.6 mA in operation, whereas the Adafruit board required 132.9 mA, or 0.492 W. LiPo batteries are available in a wide range of capacities, from 100 mAh to over 30000 mAh. Typical capacities, as they are found in smartphones and can be purchased cheaply, are around 3500 mAh, which allow a runtime of 27.6 hours. In combination with a small solar panel of around 10 Watts, continuous operation is thus easily feasible.

4.4.6 Summary

We presented Bird@Edge, an *Edge AI* system for recognizing bird species in audio recordings to support real-time biodiversity monitoring. Bird@Edge is composed of embedded edge devices, such as ESP32-based microphones and NVIDIA Jetson Nano boards, operating in a distributed system to enable efficient, continuous evaluation of soundscapes recorded in forests. We presented a deep neural network based on the EfficientNet-B3 architecture and optimized for execution on a NVIDIA Jetson Nano board to recognize bird species in soundscapes. It

[34] `https://www.msoon.com/high-voltage-power-monitor`
[35] `https://www.sparkfun.com/products/15663`

outperforms the state-of-the-art BirdNET neural network on several data sets and achieves a recognition quality of up to 95.2% mean average precision on soundscape recordings in the Marburg Open Forest, a research and teaching forest of the University of Marburg, Germany. Measurements of the power consumption of Bird@Edge Station and Bird@Edge Mics show that the system has an acceptable demand of 3.18 W plus 0.492 W for each Bird@Edge Mic, which can be covered by reasonably sized batteries and solar panels, highlighting the real-world applicability of the approach. All software and firmware components of Bird@Edge are available under open source licenses[36].

The results shown make it clear that Bird@Edge is a smart solution in the context of this thesis. The Bird@Edge classification presented in Figure 4.1 on page 26 can be justified in all three areas, i.e., computation, communication, and storage. The information analysis cost in terms of computation was reduced by the optimizations presented in Section 4.4.4, in particular the machine learning approach based on EfficientNet-B3, which was further reduced for execution on the edge device. The storage and communication costs were improved by orders of magnitude by the design of the architecture of the system, i.e., the direct processing of the audio data close to the source, since no audio data but only the labels of the classification have to be transmitted. The presented approach also improves the quality compared to established approaches for automated bird song recognition, as shown above.

[36] `https://github.com/umr-ds/BirdEdge`

5

Smart Adaptive Disruption-tolerant Networking

This chapter presents novel approaches in the area of smart adaptive peer-to-peer, delay- and disruption-tolerant networks. Particular use cases for adaptive peer-to-peer networking are disaster scenarios and emergency response applications, in which communication infrastructure might be instable, temporarily unavailable, or even destroyed.

The Serval Project is an open-source wireless ad-hoc networking system supporting a variety of communication protocols and applications. An evaluation of the delay-tolerant protocol suite *Serval Rhizome* is presented in Section 5.1.

Opportunistic named functions (ONFs) are a novel approach to operate information-centric disruption-tolerant networks (ICN-DTNs) during emergencies. Network participants specify their interests in particular content and possible application-specific functions that are executed in the network opportunistically, either partially or totally. The concept of ONFs, its Serval-based implementation, and an experimental evaluation are presented in Section 5.2.

Based on the findings of the previous section, the concept of computational offloading is further extended to the computational offloading framework OPPLOAD, presented in Section 5.3.

The modular design, implementation, and evaluation of IETF's Bundle Protocol Version 7, called DTN7, written in the Go programming language, are presented in Section 5.4.

In Section 5.5, an approach to programmable disruption-tolerant networking based on DTN7, called ProgDTN, is presented. The main idea is that network operators can incorporate context information of DTN bundles and nodes into routing algorithms that consider the specific properties of a particular application scenario.

Long range device-to-device communication can be achieved using LoRa. In Section 5.6, an approach for using LoRa with smartphones particularly for crisis scenarios is presented. The approach is integrated into DTN7 such that it allows us to adapt between different connection options.

Figure 5.1 compares the information analysis cost and achievable quality of the three main approaches (i.e., ONFs in ICN-DTNs, OPPLOAD, ProgDTN) presented in this chapter with conventional solutions. The Serval Rhizome evaluation, DTN7, and LoRa-DTN are used as inputs and/or technical prerequisites for the three main approaches. This chapter primarily investigates methods for network communication. In this domain, the improvements in terms of achievable quality relate particularly to technical metrics, i.e., QoS. The information analysis

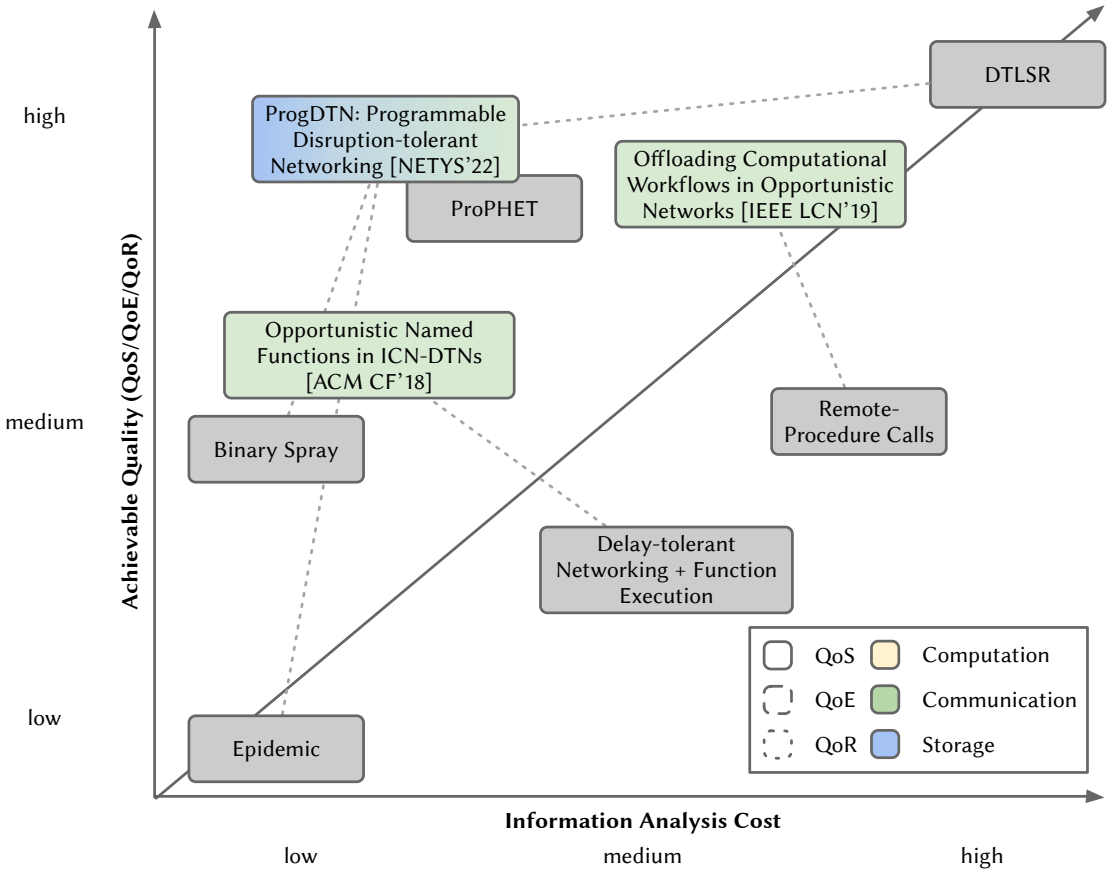


Figure 5.1: Information analysis cost and achievable quality of smart adaptive disruption-tolerant networking approaches

costs are also reflected in a particular area, namely communication. Improvements in network communication can be realized by sharing the status information of the individual participants, which, in addition to some computational cost, makes up the main part of the information analysis cost.

Many of the implementations presented in the following sections are available as open source software. The extensions of Serval regarding opportunistic functions[1], a Python interface for Serval[2], and the implementation of OPPLOAD[3] are available via GitHub. The ongoing development of DTN7 is documented on a project website[4], the ProgDTN implementation is available in the respective branch[5]. The LoRa modem firmware[6] as well as the device-to-device chat application *BlueRa*[7] are released under open source licenses as well.

[1] https://github.com/umr-ds/serval-dna/tree/nicer-hooks
[2] https://github.com/umr-ds/pyserval
[3] https://github.com/umr-ds/OPPLOAD
[4] https://dtn7.github.io
[5] https://github.com/umr-ds/dtn7-go/tree/progdtn
[6] https://github.com/umr-ds/rf95modem
[7] https://github.com/umr-ds/BlueRa

5.1 An Experimental Evaluation of Delay-Tolerant Networking with Serval

5.1.1 Introduction

The unfortunate reality is that each year disasters and emergencies in many places around the world happen, and that a common feature of these events is that partial or complete loss of communications capacity occurs. Even without the loss of communications capacity in a disaster, there are significant challenges to providing effective information for those affected [WO07]. The loss of means of communications serves to compound the difficulties and sufferings faced by those affected [CH06]. There is, therefore, a moral imperative to seek out means of finding ways to restore, or better, sustain communications during and following such adverse events.

The Serval Project [Gar11; Gar+13b; Gar+13a; Gar+12] is one of a number of endeavours that seeks to respond to this moral imperative. Its objective is to allow people to use mobile telephone handsets to communicate anywhere, anytime [Gar+13a]. The project seeks to achieve this by creating protocols, writing software, including mobile apps, and creating complementary hardware devices that, together, are able to replicate many of the functions of a conventional cellular network to some degree. The goal is not the replacement of cellular networks, but rather provisioning the best possible set of functionality and quality of service that is feasible, without requiring any conventional infrastructure.

In this section, an in-depth experimental evaluation of the delay-tolerant networking (DTN) aspects of the Serval software stack for various network setups and usage patterns, including simulated long term use, is presented. The evaluation is based on a simulation and emulation environment to provide insights into the scenarios where Serval can be deployed with satisfactory quality and performance characteristics, without requiring the expense and complication of deploying large and potentially costly physical test networks. Since battery capacity is limited on mobile phones, we take a closer at the battery drain from using Serval over various communication links, such as WiFi and Bluetooth. The contributions of this section are:

- A hybrid simulation and emulation environment is presented that allows us to run real OpenWRT[8] firmware images in an emulator, in contrast to mere simulations where only the DTN protocol can be tested.
- Various network topologies, ranging from many 1-hop neighbours and a 64-hop chain to more realistic merging islands connection schemes are evaluated.
- Several test cases mimicking common functionality, such as file distribution, messaging and peer discovery, and typical user behaviour, such as rapid bulk insertion of content, writing periodic text messages, and adding different types of content every now and then, are considered.
- Different file sizes are examined to reflect different patterns of mobile phone usage, such as sharing text files (GPX data, ebooks, messages), images (map tiles, pictures), voice and video recordings (eye-witness video footage, voice memos, diaries).

[8] https://openwrt.org/

- All test data, scripts and topologies are freely available and can be adapted to test other software[9].

Parts of this section have been published in Lars Baumgärtner, Paul Gardner-Stephen, Pablo Graubner, Jeremy Lakeman, Jonas Höchst, Patrick Lampe, Nils Schmidt, Stefan Schulz, Artur Sterz, and Bernd Freisleben. "An Experimental Evaluation of Delay-Tolerant Networking with Serval." in: *2016 IEEE Global Humanitarian Technology Conference (GHTC)*. Seattle, USA, Oct. 2016. DOI: 10.1109/GHTC.2016.7857262.

5.1.2 Related Work

There exists a wide range of related work addressing emergency communications needs and solutions, beyond what is possible to cover in this section [PK+15]. Nonetheless, many of the solutions in this space can be classified according to (1) the communications medium/media and modulation(s); and (2) the architectural model(s) used by each solution.

Communications media include WiFi, Bluetooth, WiMAX, GSM, TETRA digital radio, and various analog two-way and digital microwave, UHF, VHF and HF radio systems, as well as wired analog or digital systems, and satellite based systems, all available from various commercial vendors.

The architectural models can be often classified as either infrastructure-oriented, distributed (including peer-to-peer ad-hoc systems), or hybrid architectures of both approaches.

Several systems support multiple transport modalities. For example, WISECOM [BCW07] is an infrastructure-oriented system that seeks to provide a comprehensive approach to post-disaster communications, using satellite for global connectivity and a wide range of media and modulations. A significant challenge with such systems is their overall complexity, and their dependence on a sophisticated Internet-side infrastructure.

Distinct from the transport media, considerable work has been done on designing network protocols and frameworks for emergency communications using various selections of the media and modulations listed above [Cac+13; Wan+07; Man+14]. A resulting problem in this diversity is that interoperability can be a signifcant challenge and requires ongoing effort to contain and improve this situation [May02; Pec+15].

Mobile applications are also becoming more prominent in the emergency communications space [Gar11], due to the increasing capability of modern smartphones. Several systems also employ DTN principles to mitigate the challenges that arise when forming networks from end-user devices, and without adequate supporting infrastructure [CS14]. Such systems are particularly relevant, due of their ability to operate when faced with the failure of infrastructure, which is a common feature in disasters and emergencies [PK+15].

For example, FireChat[10] is a DTN system for sending messages, but it lacks openness. Other DTN systems such as SPAN [TRM12] and Briar[11] only support specific target operating systems such

[9] https://github.com/umr-ds/
[10] http://opengarden.com/firechat
[11] https://briarproject.org/

as Android, and SPAN does not provide applications built on top of it. Furthermore, Forban[12] can spread files opportunistically in a DTN manner, but lacks protocol support for direct private file transfers, messaging or routing.

Liu et al. [Liu+15] have developed a DTN based mobile microblogging app for censorship resistant communication. Their focus is on the app's energy consumption in an 802.11 ad-hoc network, ignoring other means of communication such as Bluetooth or WiFi in AP mode and limiting the system to specific rooted Android devices in ad-hoc networking mode. Also, there is no support for sending large files, such as videos.

Ntareme et al. [NZP11] have presented an approach based on Android phones using a store-and-forward architecture. Services such as email are transparently delivered via DTN, but the solution requires special server software in addition to the Android app. Energy and bandwidth consumption were measured, but scalability and performance in different scenarios were not evaluated.

Heimerl et al. [Hei+13] attempt to solve the problem of poor cellular coverage and power outages in rural areas by using low-cost GSM hardware and a system for reduced power consumption. While this approach is interesting for feature phones and services such as voice calls and text messages, it still requires infrastructure to function.

5.1.3 Serval

The basic concepts of Serval are presented below.

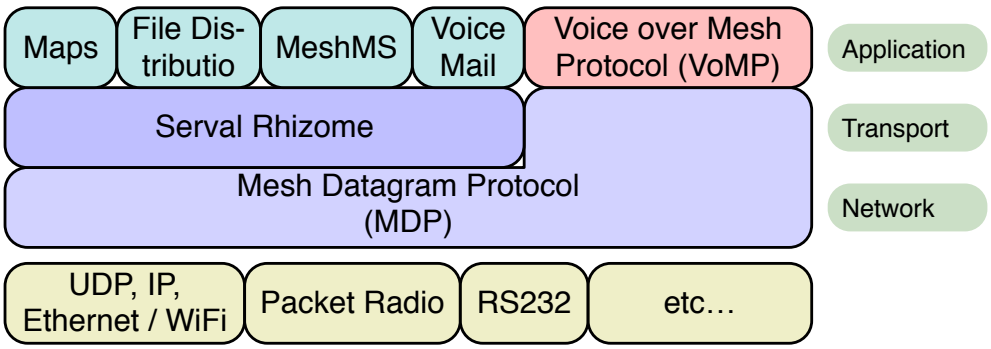


Figure 5.2: The Serval technology stack

Overview

Serval is centered around a suite of protocols and technologies designed to allow ad-hoc infrastructure-independent communications [Gar11; Gar+13b], as illustrated in Fig. 5.2. The goal is to provide infrastructure-independent versions of many of the services that are commonly used on smartphones in conjunction with the Internet and/or cellular networks, e.g., voice calls, short text messaging (SMS), voice mail, social media, as well as file and image transfer.

[12] http://www.foo.be/forban/

The Serval Mesh protocols purposely take a contrasting approach to that of using IP (v4 or v6) as the basis for forming mobile ad-hoc communications networks (MANETs) [Gar+13a]. The reason for this is that despite billions of dollars of research and development work, IP-based MANETs still struggle, and face a number of significant challenges that limit their real-world use, e.g., address allocation, the need to maintain a routing table, authenticity and integrity of communications, and the need for relatively reliable and stable end-to-end connectivity for such systems. Instead, Serval uses 256-bit public cryptographic keys as the primary network identifier, the so-called Serval ID (SID), and also includes a rich security model that facilitates confidentiality, integrity and authenticity by design, and does not require a Trusted Third Party (TTP) to operate. It also includes a store-and-forward DTN protocol (Rhizome), allowing network operation in the absence of end-to-end connectivity.

Rhizome and Delay Tolerant Networking

Rhizome is a simple bundle protocol that principally defines data units as *bundles*, consisting of an optional payload, together with a *manifest* that contains necessary meta-data. *Manifests* have a hard size limit of 1 KB to improve efficiency, and must also contain a cryptographic public key that is used to protect the integrity and authenticity of the manifest itself. The manifest may also contain a cryptographic hash, indicating that it has an associated payload, together with other meta-data, such as mime-type, Rhizome service tag, file-name, and SID of the sender and/or recipient, as appropriate.

While the Rhizome implementation includes several transports for Rhizome, including HTTP, packet radio and the Serval MDP protocol described below, the protocol is purposely agnostic of the transport, to allow other transports to be added. The intention of this is that any transport that is capable of carrying bytes of data can be used to transport Rhizome data.

As a simple state-less flooding protocol, Rhizome requires no routing table, and never requires that two parties have an end-to-end connection for them to communicate. That is, the Rhizome protocol is always focused on single-hop communications, with multi-hop communications emerging as a natural consequence of *bundles* replicating among nodes.

Rhizome is used as the basis for the SMS-like Mesh Messaging Service (MeshMS) [Gar+12], and file distribution, including software updates. It is also planned to implement a twitter-like micro-blogging service using Rhizome.

MDP, MSP and Node Discovery

In addition to the Rhizome DTN protocol, Serval also includes a real-time packet-switched protocol, the Mesh Datagram Protocol (MDP) that is generally similar to UDP/IP, but uses SIDs instead of IP addresses, and includes encryption, authentication and integrity features by default. The TCP-like Mesh Streaming Protocol (MSP) is layered atop MDP to provide reliable data streaming. Various services can be implemented atop MDP and MSP, including the VoIP-like Voice over MDP Protocol (VoMP).

MDP routing uses an OSLR- and BATMAN-inspired [JNA08], [Jac+01] ad-hoc protocol for both node discovery and maintaining a routing table, that facilitates multi-hop routing of packets.

In order to reduce packet sizes, address abbreviation is used, so that only the minimum number of bytes of a SID is required to uniquely identify a node among its direct, i.e., 1-hop neighbours. This reduces the header size in the common case to be smaller than that used for IPv6.

5.1.4 Experimental Evaluation

The experimental setup for our in-depth evaluation of Serval, including the hard-/software environment used, the parameters measured, the network topologies chosen, and communication scenarios, is presented below.

Simulation/Emulation Environment

To evaluate the performance in a realistic manner, we have developed a simulation/emulation system called *MiniWorld* based on QEMU/kvm and Linux networking code to achieve full system emulation. This gives us the opportunity to use the OpenWRT build chain for building router images that include Serval. OpenWRT is also used on real world routers such as TP-Link MR3020 or the Mesh Extenders of the Serval Project. Having a full operating system with its own network stack running on each node gives a much better picture of real life performance than pure protocol simulation.

All tests are performed on a 64 core AMD Opteron 6376 CPU with 256 GB RAM, simulating up to 100 virtual nodes, each one with 512 MB RAM and 2 GB of storage space. These quite limited values allow us to investigate how Serval performs on older smartphones like the original Samsung Galaxy S or similar, which are common in developing countries.

Measurements

Standard Unix tools are used to measure system properties, with a measuring interval of one second. For memory consumption, CPU and I/O usage *pidstat*[13] is used to monitor the statistics of the Serval process from within a node. Disk space is measured with *du* and *df*, both from the GNU coreutils[14]. Network usage is measured on the *MiniWorld* bridge interfaces of the host system using a custom Python tool[15] based on *libpcap*[16]. Insertion points in time for the Rhizome store are derived directly from Serval's log, while the general file count is logged using direct *servald* calls.

[13] http://sebastien.godard.pagesperso-orange.fr
[14] http://www.gnu.org/s/coreutils/
[15] https://github.com/umr-ds/serval-tests/blob/master/netmon.py
[16] http://www.tcpdump.org

Network Topologies

Several network topologies are studied, as shown in Table 5.1. The *Hub* topology connects 48 nodes with each other. It represents a scenario with a high number of direct neighbours all using bandwidth, flooding each other with status information and new files, sharing the same transport channel. Typically, the number of direct neighbours is limited by the radio range of WiFi or Bluetooth (i.e., often less than 48). Thus, *Hub* is challenging for Serval and also the radio link itself.

Name	# Nodes	Description
Hub	48	All nodes connected to each other
Chained	64	Pair-wise connected
Islands	100	Partitioned islands, merging over time

Table 5.1: Topologies

The *Chained* topology consists of a chain of 64 nodes, thus the last node is 63 hops away from the first node. Typically, network connections over the Internet require less than 16 hops. In a delay-tolerant mobile mesh network, more hops might be needed for messages to reach their destination compared to static networks physically optimized for minimum hop numbers and maximum throughput.

The *Islands* topology represents a partitioned network that slowly merges over time. At the beginning, there are 100 nodes in small islands with only a few neighbours. Between these small islands there are no links, but after a predefined time a few of them merge together, exchanging all their information that they have collected so far. Finally, there are two big islands where one node acts as a bridge between the two, and all accumulated data from one island has to pass through this node to propagate to the other island.

All topologies are used in two configurations, one modeled after the common 802.11g standard with a 54 Mbit/s limit on each link and one with no bandwidth limitations.

Scenario Tests

Based on these topologies, we designed several tests, as shown in Table 5.2.

Idle (I) simply starts Serval and waits until all nodes have found each other. This test serves to evaluate how long the discovery phase takes in various network setups and how much traffic Serval produces while idling.

Mass Files (MF) pre-generates a number of files and inserts them at one specific node. The goal is to evaluate whether Serval can handle a large number of files at once. Propagation through the network is observed to reveal problems related to high bandwidth, storage and/or CPU usage.

Name	Short	Description
Idle	I	Node discovery, no actions triggered
Mass Files	MF	Insert bulk of file set at once
Mass Messages	MM	Insert bulk of messages at once
Periodic Files	PF	Periodic adding of files
Periodic Private Files	PPF	Periodic adding of private files
Periodic Messages	PM	Periodic sending of messages
Combined	C	All periodic tests together

Table 5.2: Scenario Tests

Mass Messages (MM) is designed to test the messaging subsystem of Serval by flooding the network with text messages. A number of messages is sent at once to every single node in the network no matter if it is currently reachable or not.

Periodic Files (PF) is designed to observe the long-term behaviour of the system. Files are added at random points in time by every node. A real world analogy is: people taking pictures occasionally and sharing them with everybody else.

Periodic Private Files (PPF) is a special case of *PF* where files are not shared with the public but sent to a randomly chosen recipient.

Periodic Messages (PM) is designed to evaluate the Serval messaging subsystem. These messages are also directed to a specific recipient and are not meant for the public.

Combined (C) is designed to run all periodic tests (*PF, PPF, PM*) at once. Similar to real life situations, the nodes change their behaviour and there is a competition for the resources in the network. Broadcasting files, sending files to "friends" and writing text messages all have different requirements.

Data Sent

Text messages consist of a fixed string plus a timestamp in milliseconds when a message was sent. Since these messages are meant to mimic real world chat, the total string length is kept small (53 characters). According to a chat study of Battestini et al. [BSS10], text messages sent by males had an average length of 47 characters and for females 58 characters.

Files have different file sizes representing different types of data, as shown in Table 5.3. The *Small* file set contains randomly generated files ranging from 64 KB to 512 KB; large text files, ebooks, small pictures or other data such as map tiles typically have these sizes. In the *Medium* file set we have files between 1 MB and 10 MB, which is nowadays the size of pictures taken with mobile phones or some audio recordings. Recorded video or software bundles are represented in the *Large* file set and are generated in the range from 25 MB to 100 MB. Finally, there is a *Mixed* file set where small, medium and large files are included.

Name	Sizes	Description
Small	64K, 256K, 512K	Small pictures, map data, text files
Medium	1M, 5M, 10M	Camera pictures, audio recordings
Large	25M, 50M, 100M	Recorded video
Mixed	all of the above	-

Table 5.3: Test File Sets

Test Execution

All file related tests were performed with all four file sets, every test was executed on all topologies with limited and unlimited bandwidth resulting in a total of 114 tests. While some tests (e.g., *MF*) are count-based and terminate after every node has received a specific number of files, other tests (e.g., *PPF*) are time-based - always running for the same duration. We performed 5 iterations of each test, resulting in a total of 570 test runs.

Experimental Results

In this section, various results regarding Serval's behaviour during the experiments are presented.

Idle Behaviour

To investigate the idle behaviour of Serval, we looked at network traffic, CPU load and memory usage after the initial discovery phase, without triggering further actions. In every scenario, whenever Serval is started, there are peaks in the network load, in the *Chained* and *Hub* topologies at approximately 10 to 12 Mbit/s. After this peak, *Chained* has a summed average network traffic of around 0.7 Mbit/s, whereas the nodes in *Hub* produce 6 Mbit/s. This behaviour is caused by Serval's information distribution strategy, because it announces status information, such as the list of files in Rhizome, periodically via broadcasts. Since there are 47 neighbours for each node, traffic is relatively high in the *Hub* topology. *Islands* has extrema whenever partitions merge. The traffic during peaks grows with the number of nodes.

CPU usage of the Serval process correlates with network load in our scenarios, but never gets larger than two percent per node. Serval uses around 4 MB of memory in all scenarios.

Moreover, the discovery time of each topology is different. For *Hub*, the average time of a full network discovery is approximately 5 seconds, since every node has a direct connection to all others. In contrast, the *Chained* topology takes about 20 seconds, because announcements have to be forwarded through all other nodes.

In some experiments, Serval's address abbreviation (Sec. 5.1.3) mechanism caused conflicts under special circumstances, depending on the keys and when different nodes announce

themselves for the first time. If a node already has seen another node with the same abbreviated address, it is ignored, potentially causing a partitioning of the network. To circumvent such effects, we modified Serval to generate unique prefixes for the desired node number in our tests.

Hub Constraints

For *Hub*, a single bridge interface was used to connect all nodes. Since each node is a single hop away from all other nodes and Serval uses broadcast packets to announce meta-data (e.g., the files of a node), each node is flooding all neighbours with this information. Since the number of adjacent nodes affect the CPU consumption of the respective node, in the *Hub* topology the CPU usage is always higher than in the corresponding test in *Chained* or *Islands*, due to the high number of direct neighbours.

Topology Characteristics

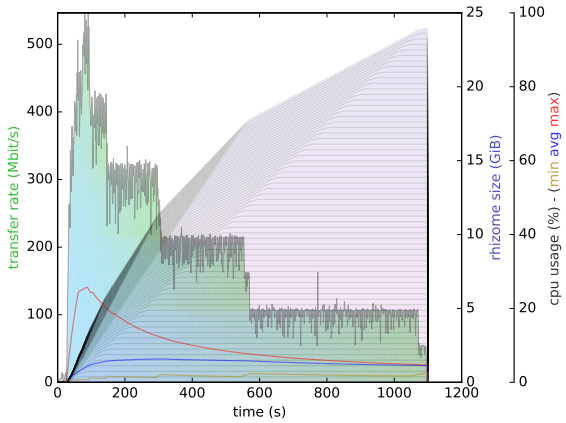
Fig. 5.3 shows *Mass Files* tests with a *Mixed* file set in different topologies. It shows how transfer rate in Mbit/s, size of the Rhizome database and the CPU usage change over time. The transfer rate is stacked for all links. The Rhizome size is the stacked database sizes of all nodes.

Fig. 5.3a shows a limited (802.11g) *Chained* topology, in which five phases are visible, caused by the Rhizome prioritization based on file sizes. Small files are delivered first and therefore can be distributed earlier by the following nodes. The bigger the files get, the less total network utilization is achieved. Despite this effect, a constant stable data flow is visible, and the Rhizome store grows constantly. The maximum CPU load correlates with network usage, since the most active network nodes do have the highest CPU usage.

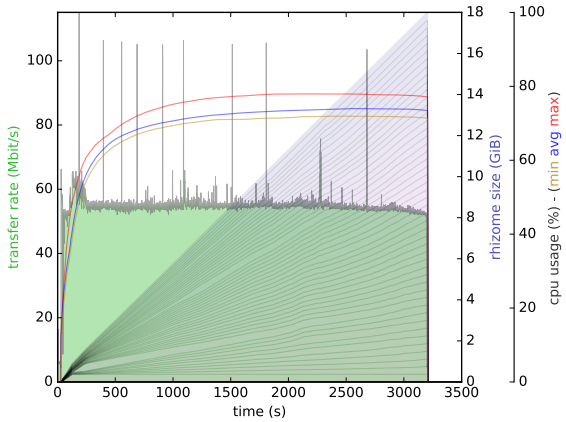
In Fig. 5.3b, a limited *Hub* topology is shown. Though a constant 54 Mbit/s data flow is visible, the spikes exceeding 54 Mbit/s are measurement errors, caused by differing network backend and traffic monitoring timers. With a constant network load caused by the file transfers, the disk usage also grows linearly as expected in this case, meaning that the network load is not dominated by status and management information but real content distribution. Compared to Fig. 5.3a the average CPU usage is about 10 times higher, as explained in Sec. 5.1.4.

For *Islands*, CPU usage increases every time the network changes. Looking at *Periodic Files* tests, the max. CPU load rises to 15% when large files are inserted, since they have to be redistributed among the other nodes. Fig. 5.3c shows the *Mixed* file set in *MF*, which peaks at around 7% CPU load. Since many of the files already exist on various nodes, every time new network connections are set up, the impact on the CPU is relatively low compared to *Hub*. In general, smaller files have a negligible impact on the CPU.

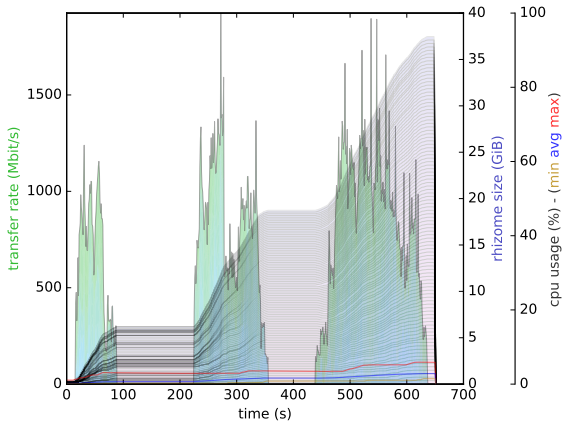
The *Periodic File* tests with small sizes do not show any unexpected behaviour in terms of CPU consumption in *Chained*, the CPU peaks at about 10%. When the files are encrypted as in *PPF*, the CPU utilization is slightly higher, at about 15%, due to CPU intensive cryptographic operations.



a) *Chained*



b) *Hub*



c) *Islands*

Figure 5.3: *MF Mixed*: Cumulated Rhizome store size, network and CPU load.

The file size influences CPU utilization, which greatly impacts the inserting node. For instance, when sending *Small* files in *Chained*, there is no significant change of CPU utilization compared to idling, whereas file set *Large* utilizes the CPU up to 35%. Bigger files lead to more time

consuming hashing, as it is required by the corresponding protocol. Thus, every node receiving the file needs to compute a hash, verify and redistribute it, which also leads to a higher load.

In terms of CPU usage, *Islands* is a combination of *Chained* and *Hub*. CPU usage does not exceed 50%, since the total number of neighbours per node is not as high as in *Hub*.

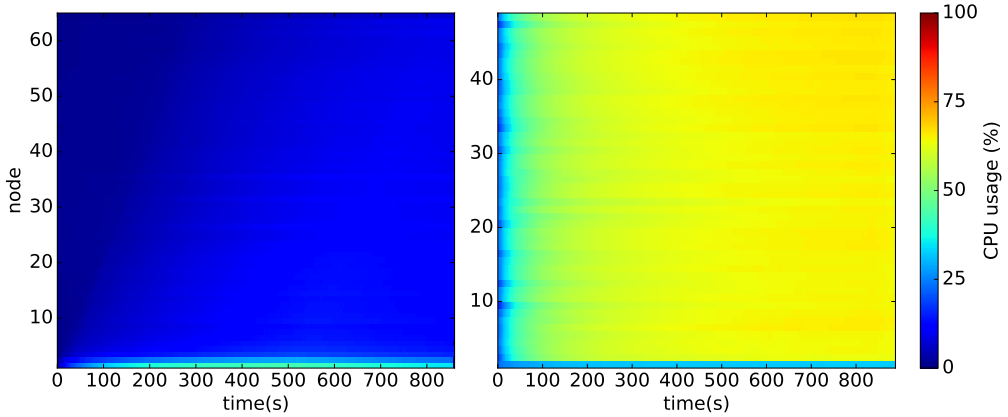


Figure 5.4: *MM* CPU usage over time. Left: unlimited *Chained*, right: unlimited *Hub*.

In the message based tests, the measured CPU consumption correlates with the number of messages sent. For *MM*, the behaviour differs depending on the topology used. Fig. 5.4 shows the CPU usage per node of two experiments over time. Using *Chained*, the inserting node peaks at 30% CPU load compared to the receiving nodes, which consume about 15%. Using *Hub*, the load of the inserting node remains the same. In contrast, the receiving nodes constantly consume about 65% CPU. *Hub* suffers from the broadcast overhead (Section 5.1.4), but this does not fully explain the high load, as the sending node is not affected. Further investigating this behaviour, we tracked it back to recurring hashing and encryption in Rhizome Journal syncing, which is the core of MeshMS messaging.

PM results differ from *MM*. For *Chained*, the CPU utilization is relatively low at about 15% maximum. This correlates with the CPU load of the non-inserting nodes in *MM*. Since they are added periodically, the CPU overhead is negligible here. *Hub* behaves differently than in the file based tests or *MM*: The *PF* tests show that in every topology the more files are injected in the network, the more CPU is needed to handle the broadcast packets. Messages are not announced further after reaching their destination and being acknowledged by the recipient. The obvious consequence should be that the CPU usage decreases. However, as indicated by Fig. 5.5, once the CPU peaks at about 25%, it does not settle any more, but increases even further, although the network load decreases to the idle level and the Rhizome database size is at its maximum, which indicates that all messages have arrived. This behaviour cannot be transferred to *Islands*, where the inserting nodes peak at about 40% and all other nodes do not exceed 15%.

For *C* tests, the general CPU usage is similar to other file based tests. The only difference is the fact that in *Chained* and *Hub* the CPU usage increases by 5% after about 500 seconds and also correlates with the network load, similar to the behaviour depicted in Fig. 5.5. This problem emerges when sending messages over a longer time period. Since *Islands* is not in the final

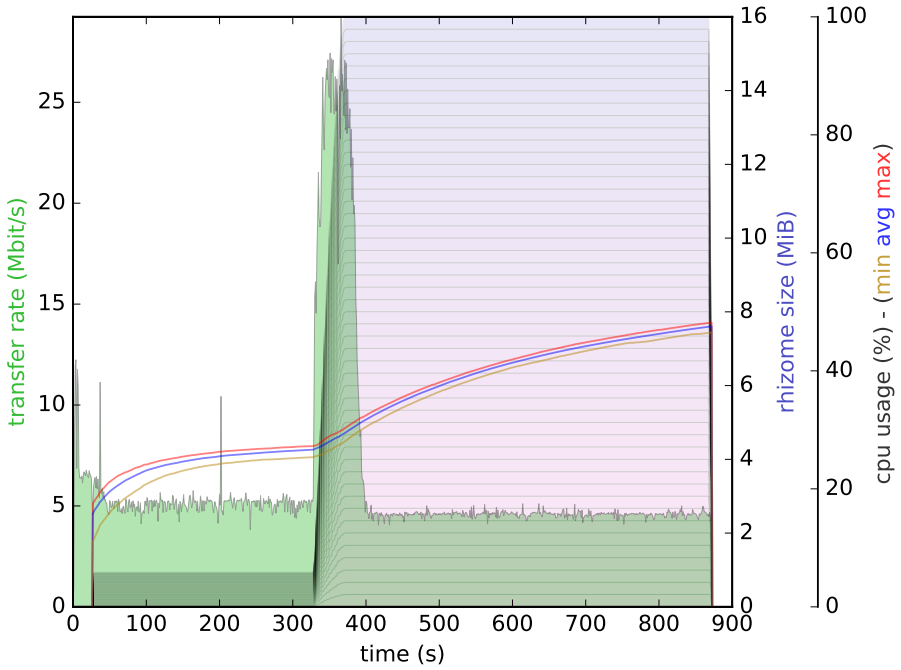


Figure 5.5: *Hub* limited *PM*: Rhizome store size, network & CPU

state at the beginning of the test in terms of the links between the nodes, this result can not be observed in this particular topology.

Network Performance

One goal was to test to what extent Serval is able to use available bandwidth. *Chained* was created to assess this.

The cumulative transfer rate using Rhizome in this topology reached 500 Mbit/s to 2 Gbit/s, depending on the file sets, with *Large* being the fastest. That is, up to 2 Gbit/s of traffic was being carried over the set of hops in the chain, with each seeing an average utilization of 32 Mbit/s. Tests that transfer large files over an unlimited network show that Serval is able to use even more bandwidth, since the highest measured transmission speed from one node to another can be up to 160 Mbit/s.

Using *Chained*, the hop-to-hop transmission time can be modeled, since node n is able to receive a file just after node $n-1$ received it. Fig. 5.6 shows the hop-to-hop transmission times of the *Medium* file set. The five files of each size are grouped into one box plot, while the colors present five different runs of each experiment. The median transmission times for 1, 5 and 10 MB files are 0.54, 1.06 and 1.85 seconds, and only 0.27 sec for 64 KB files. From these values, a simple correlation for the transmission time can be derived: $T(size_{MB}) = 0.16 \cdot size + 0.26$, which also holds for the *Large* set. The formula indicates a net transmission rate of around 31 Mbit/s, with a 0.26 sec delay.

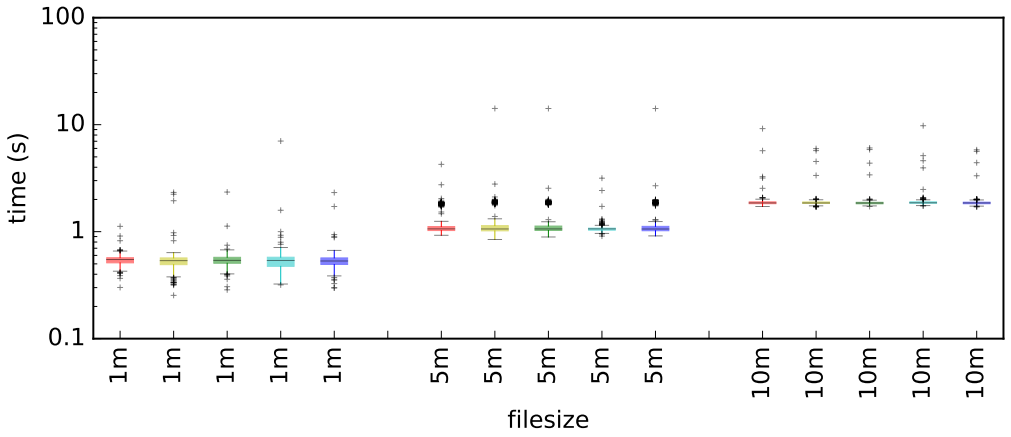


Figure 5.6: *Chained* limited *Medium* file set: File-size-grouped hop-to-hop delivery periods of five runs.

The average speeds are lower, because files are exchanged node-by-node, and can only be spread to node $n + 1$ after reaching node n, resulting in an effective end-to-end bandwidth, for a given bundle, inversely proportional to the number of hops. This compares favourably with end-to-end ad-hoc wireless routing protocols, where the effective end-to-end bandwidth drops by approximately half for each additional hop.

Briefly considering the different topologies, the network utilization in *Islands* for file based tests is generally about the same as in *Chained*, since each node has only a few neighbours, in contrast to *Hub*, which is always able to saturate all links due to the high degree of connection among nodes.

Messages in Serval are effectively transported as small files, with a payload size of 53 bytes in both *PM* and *MM* cases. The network load shows a behaviour similar to small files in the *PF* test, peaking at up to 40 Mbit/s at all topologies and regardless if the network is limited or not.

The network load for *C* tests in all topologies is similar to the file based tests, independent of bandwidth limitations. The only difference is the increase of the network load after about 500s on *Chained* and *Hub*, as shown in Section 5.1.4.

In *Hub*, small files take between 1 and 4 min to arrive on the last node in the limited network links. This increases linearly, up to 20 min, with increasing file size. If the network is unlimited, transmission time reduces to between 18 sec and 9 min, depending on the file size. One difference between *Hub* and *Chained* is the runtime. *Small* files are transmitted faster in *Hub*, whereas *Large* files are faster in *Chained*. The time overhead for file announcements is relatively higher for *Small*. Even with a lower total bandwidth (*Hub*: 54 Mbit/s for 48 nodes vs. *Chained*: 54 Mbit/s pairwise), *Hub* can achieve faster transmission rates. The limitation of network speed does not influence this behaviour, only the overall transmission time increases.

The transfer times of messages differ from topology to topology. While it takes about 350 sec in *Chained* until all messages arrive at their destinations, it can take up to 900 sec in *Hub*. This again shows that the high number of 1-hop neighbours in *Hub* is challenging for Serval. The

transmission time for messages in the *C* tests depends highly on the used file set, rather than on the topology and network speed. The reason is that the network is saturated with big files, which leads to overall higher transmission times for messages.

Energy Consumption

The *Idle* test in Section 5.1.4 showed network peaks caused by Rhizome status information announcements. Therefore, the energy consumption of the announcements is evaluated: Two devices send announcements in different intervals. Fig. 5.7 shows the energy consumption for peer A using different announcement intervals at peer A and peer B. With a 0.5 sec or 1 sec interval, the consumed energy is 9% higher than in idle state. With a 2 sec interval, the consumed energy is only 3% higher than in idle state. With a higher interval of 4 sec or 8 sec, only negligible decreases in energy can be achieved.

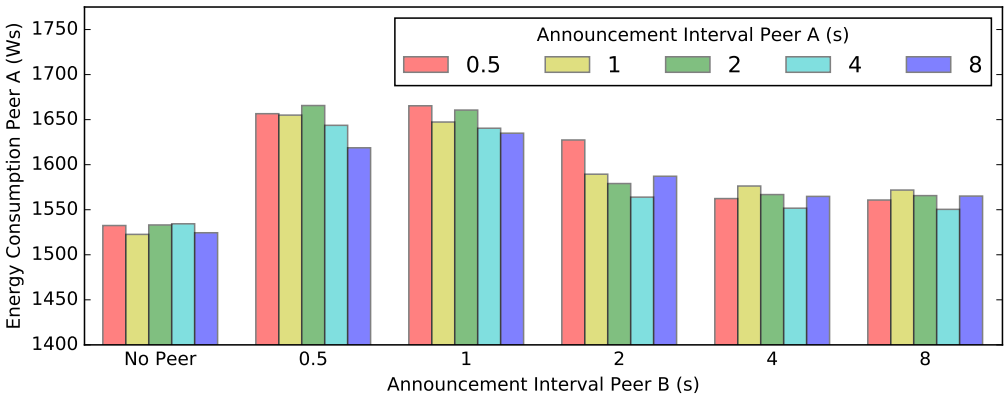


Figure 5.7: Energy consumption of announcement intervals

Furthermore, the power consumption during *MM* and *MF* tests were evaluated. Two peers were connected via an 802.11n WiFi Access Point. Peer A inserts files and messages into Rhizome in the same manner as *MM* and *MF* tests. The power consumption of peer B, a Raspberry Pi 3, is then measured with the Odroid Smart Power measurement device, an external power meter. The aim of these experiments is to measure the energy overhead for running Serval on a device, which allows conclusions about the power drain of Serval on battery-powered devices.

Fig. 5.8 shows the power consumption during different Rhizome file set insertions similar to *MF*. The file sizes are increased during the phases *f1-f4*. During *f1*, the file sizes are smaller than 1 MB, resulting in a negligible additional power consumption. The bigger the transmitted files are, the more power is consumed. The comparison between receiving files and sending files shows an unexpected behaviour: In all phases *f1-f4*, sending files is less expensive compared to receiving files, on the average between 0.05 and 0.1 W (3-6%). This counterintuitive result is caused by additional CPU consumption of the Rhizome checksum calculation during reception. Compared to a 1.53 W mean idle value of peer B, the power overhead introduced by Serval is between 0.01 and 0.13 W (1-8%) during phases *f1-f4*.

In another experiment, we measured the power consumption during different message insertions similar to the *Mass Messages* test. The results show a power consumption peak between

1.81 and 1.91 W during a short period of reception, followed by a phase of negligible additional power consumption. During the reception of 100 messages, a mean value of 1.69 W (10%) additional power consumption is measured.

A better energy efficiency during message transmission could be achieved by using Bluetooth. It consumes a significant amount of energy during device discovery, but has a lower power consumption during data transmission than WiFi. Due to the low energy efficiency (joule per bit) of Bluetooth compared to WiFi, it consumes significantly more energy for large data transmissions. During an experiment, we measured a 32 times better energy efficiency of WiFi compared to Bluetooth for files between 512 KB and 16 MB.

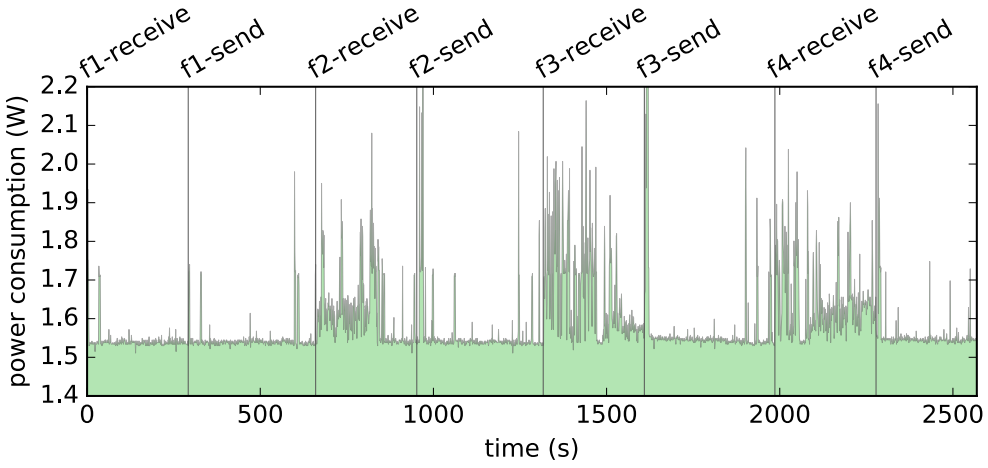


Figure 5.8: Power consumption during different Rhizome file set insertions (*f1-f4*) similar to the *Mass Messages* test.

5.1.5 Summary

In this section, an in-depth experimental evaluation of the delay-tolerant aspects of Serval for various network setups and usage patterns, was presented. The results show satisfactory performance of Serval when deployed in partitioned scenarios and extreme examples of network topologies. Furthermore, we have analyzed Serval's energy consumption, having the limited battery capacity of mobile devices in mind.

In particular, our experiments indicate that there is a sweet-spot for the trade-off between up-to-dateness and energy consumption regarding announcement intervals. Furthermore, Serval can handle a realistic number of files over a longer time period. In the *Chained* topology, neither the CPU load nor the used network bandwidth leads to out of service situations. All tests with the *Hub* topology show that in a highly used network the announcements consume a considerable portion of the available bandwidth. In emergency situations or in long-term setups this could have a negative effect depending on the number of people in direct communication range. The *Combined* tests in our *Islands* topology demonstrate that Serval works flawlessly in adapting to heterogenous environments where users have different requirements at the same time and the topology changes over time.

5.2 Opportunistic Named Functions in Disruption-tolerant Emergency Networks

5.2.1 Introduction

Facilities like power stations, water reservoirs, and telecommunication centers are part of the critical infrastructures that are vital for modern societies. During and in the aftermath of a disaster or an emergency event, such as an earthquake or a terrorist attack, it is essential to maintain these infrastructures and to restore and to repair capacities that have been damaged. Telephone lines, cellular base stations and parts of the Internet backbone might be destroyed, disrupted or overloaded due to network congestion. However, information about the current situation is crucial for affected people and rescue teams. Thus, it is important to re-establish basic means of communication during a disaster despite fragmented IP networks and totally or temporarily disrupted network links.

In the past, mobile ad-hoc networks (MANETs) and disruption-tolerant networks (DTNs) were studied as approaches to re-establish basic communication services during disasters. More recently, information-centric network (ICN) protocols for disaster scenarios, such as ICN-MANETs [OLG10] and ICN-DTNs [Mon+14; Che+16a; Ana+16] were proposed. Apart from important challenges such as authentication and access control, ICN protocols address the following aspects in a disaster scenario: (i) since end-to-end connectivity is not guaranteed and location-based as well as fixed addresses may not work, name resolution at the network layer (instead of at the application-layer) can be used, which also supports anchor-less mobility, (ii) since caching, traffic engineering and prioritization based on the name of the desired content is an inherent concept of ICNs, network nodes can make a trade-off between, for example, bandwidth and storage based on the relevance of the content for a consumer. Furthermore, Named Function Networking (NFN) [TS14] as a generalization of Named Data Networking (NDN) [Jac+09] offers great potential for providing support in disaster scenarios. In NFN, names do not only refer to data, but also to functions and computational tasks, and the network's role is to resolve names to computations.

In this section, *Opportunistic Named Functions* (ONFs), a novel approach to extend the NFN paradigm for DTNs in disaster scenarios, are presented. ONFs are *opportunistic* in the sense that (i) named functions are applied based on locally optimal decisions using criteria such as network congestion avoidance, battery lifetime and device capabilities, and (ii) named functions are applied after the receipt of data transmitted during opportunistic communication. In particular, the section makes the following contributions: (1) We present a novel approach for ICN-DTNs in which ONFs are used for data preprocessing, analysis, integration and transfer within wireless networks in disaster scenarios where in-network processing can provide essential information for situation analysis. (2) We introduce a novel implementation of ONFs within Serval, a well evaluated open-source DTN project for disaster situations [Bau+16]. (3) We propose novel methods for image (pre-)processing and face detection in disaster scenarios to support the search for missing persons. We further present experimental evaluations with respect to battery consumption and runtime of these methods. (4) We present simulations for using ONFs in basic network topologies and a disaster scenario.

Parts of this section have been published in Pablo Graubner, Patrick Lampe, Jonas Höchst, Lars Baumgärtner, Mira Mezini, and Bernd Freisleben. "Opportunistic Named Functions in Disruption-tolerant Emergency Networks." in: *ACM International Conference on Computing Frontiers 2018 (ACM CF 2018)*. Ischia, Italy: ACM, May 2018. DOI: 10.1145/3203217.3203234 .

5.2.2 Related Work

ICN-DTNs were 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, WiFi, TETRA digital radio or satellite links) introduce more complex topologies of heterogenous 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 with at a low data rate.

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 were applied by Ashar et al. [Ash+16] and 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 (ONFs).

Melvix et al. [JLP15] 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 then the last 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.

5.2.3 Opportunistic Named Functions

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.

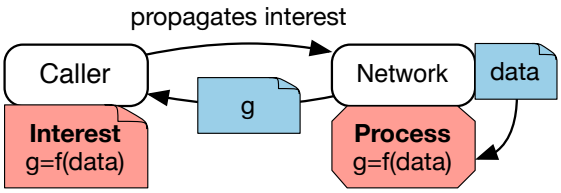


Figure 5.9: Basic ONF concept

To address these problems, we propose *opportunistic* named functions (ONFs). Figure 5.9 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 5.10 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 5.10, 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

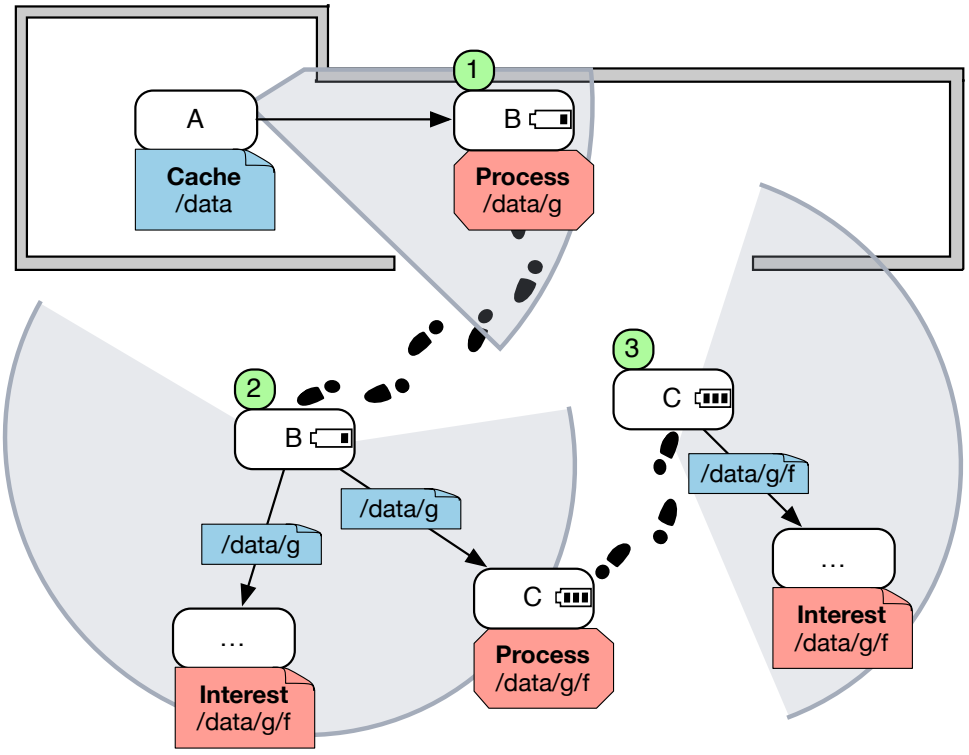


Figure 5.10: Example of in-network processing of named content with ONFs

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.

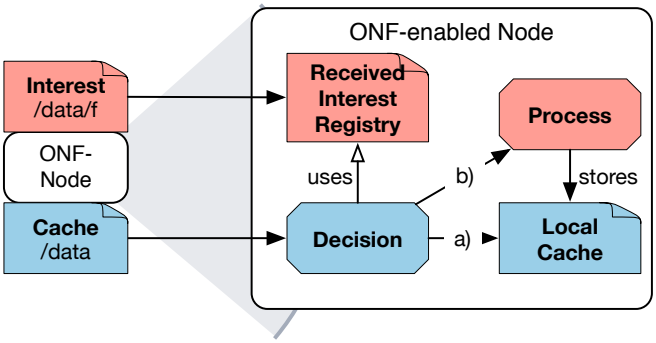


Figure 5.11: Functionality of ONFs

Figure 5.11 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.

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 an 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 us 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 5.2.5.

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.

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 mobile phone's camera, 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 5.2.3, 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 us 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.

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.

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

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 ← longestPrefixMatch(functions);
4      return interests.functions;
5  Function getPriorityMatches(parsedFunctions, functions)
6      for f in parsedFunctions do
7          matches ← priority of f in functions;
8      return matches;
9  Function getBestMatch(interests, matches)
10     result = ∅;
11     sorted_matches = sort matches: m1 > m2 if m1 has more interests than m2;
12     for m in sorted sorted_matches do
13         result ← result ∪ m.function;
14         if all interests applied then
15             break;
16     return result;
17 F' ← get applicable functions from F;
18 for not F'.isEmpty() do
19     I' ← get resolvable interests for name in I;
20     functions ← parseFunction(I', F');
21     matches ← getPriorityMatches(functions, F');
22     M ← getBestMatch(matches);
23     name ← apply each function in M on name;
24     F' ← get applicable functions from F;
```

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.

5.2.4 Opportunistic Named Functions in Disaster Scenarios

In a disaster scenario, the search for missing persons is a primary task for rescue teams and highly important for affected people. In such situations, sensors, battery-powered mobile devices, and generator-powered routers remaining from the communication infrastructure can be used to establish a resilient disaster-response communication system. The inherent tradeoffs between either spending CPU cycles and battery power for ONF function execution, or spending battery power and air time for data transmissions, is illustrated by the following example. A content consumer is interested in all detected faces in all pictures taken by the mobile devices' cameras. Since state-of-the-art mobile phone cameras usually provide a resolution of 8–16 megapixels, simply transferring all raw images from the content producer to the consumer has the risk of producing network congestion. In contrast, ONFs can be 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 our scenario, ONFs are used to support the search for missing people in image files that are distributed via an ICN-DTN. Figure 5.12 illustrates the processing of ONFs in an ICN-DTN within this disaster scenario. 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. This is reflected by the specified interests in content, which in this case are also formulated with domain knowledge about the applicable filters, scalings, and lossy compressions. In a real-world scenario, this process could be transparently handled by an additional application provided to rescue teams. In our scenario, we have built a small software component that relies on detected faces, either in their original color or transformed to a grayscale image, and low-resolution black/white images. The component visualizes the information by showing detected faces in high resolution, with a low-resolution black/white-underlay for situation analysis. It is not sufficient to transfer the low-resolution part only, since it is not possible to perform face detection algorithms on black/white images with accurate results [Lam+17].

In Figure 5.12, we assume 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 that is able to make basic image transformations, such as grayscaling and shrinking a black/white version of an image. In our case, 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 grayscaled image. It extracts 3 faces (0.1 MB, 0.23 MB and 0.12 MB) that are then transmitted via 802.11 WiFi. At the content consumer, the scaled black/white image and each of the three detected faces arrive separately. The software component on top of the ICN-DTN is notified at each arrival and provides an image to the user that is successively completed with a low-resolution black/white image in the background and grayscale faces in the foreground, which can be used for situation-awareness.

To summarize, this scenario illustrates the operation of ONFs in ICN-DTNs (i) without end-to-end connectivity (ii) with unpredictable routes, and (iii) with heterogenous devices on the route through the network.

5.2.5 Implementation

In our disaster scenario, ONFs are used to support the search for missing persons through image files that are distributed via an ICN-DTN. Therefore, 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, as well as the applied functions, are described below. Image transformation and face detection functions are introduced that can be used to trade between execution time, resource consumption and quality of results.

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 Hooks [17]. 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 the RIR, a name resolution component, and a function execution environment. Functions are implemented as executable binaries that reside either

[17] source code available at `https://github.com/umr-ds/serval-dna/tree/nicer-hooks`

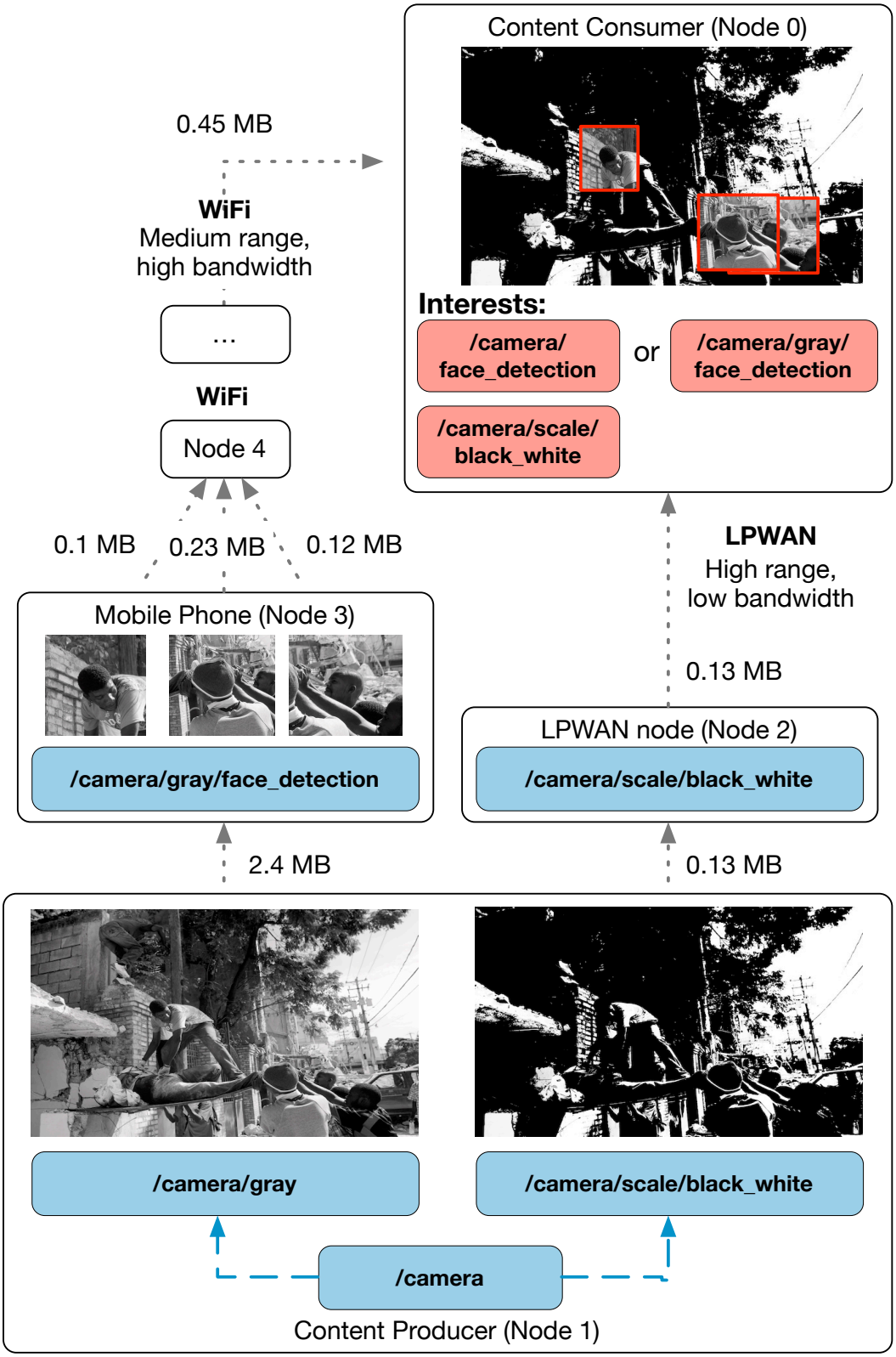


Figure 5.12: Processing ONFs in a disaster scenario

(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.

Implemented ONFs

The implemented ONFs for our scenario are described below.

Image (Pre-)Processing

For basic image preprocessing purposes, we use the OpenCV[18] 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.

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 5.13 illustrates this property by showing regions of high interest for relevant topics with red color and

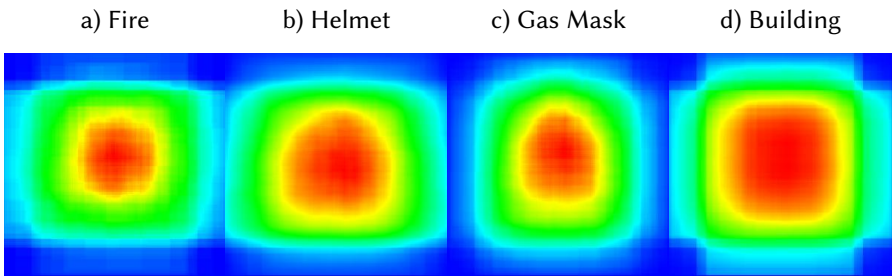


Figure 5.13: Regions of interest in photos for relevant topics

[18] http://opencv.org

regions of low interest with blue color in a heatmap. Therefore, heatmaps were generated from bounding boxes provided by the ImageNet[19] image library that organizes images according to the WordNet[20] hierarchy, i.e., according to meaningful concepts, described by multiple words or word phrases. For each topic, a matrix with 256×256 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.

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*[21] is used to verify the 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. Additonally, *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.

5.2.6 Experimental Evaluation

We present an experimental evaluation of ONFs below. In Section 5.2.6, image (pre-)processing, smart image fragmentation, and face detection are evaluated in terms of runtime and power consumption. In Section 5.2.6, these experimental results are used to simulate ONFs in different basic topologies as well as in a disaster scenario.

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 (RPi). This device is used as

[19] http://image-net.org
[20] http://wordnet.princeton.edu/
[21] http://dlib.net

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 ODROID 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 differs 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 × 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.

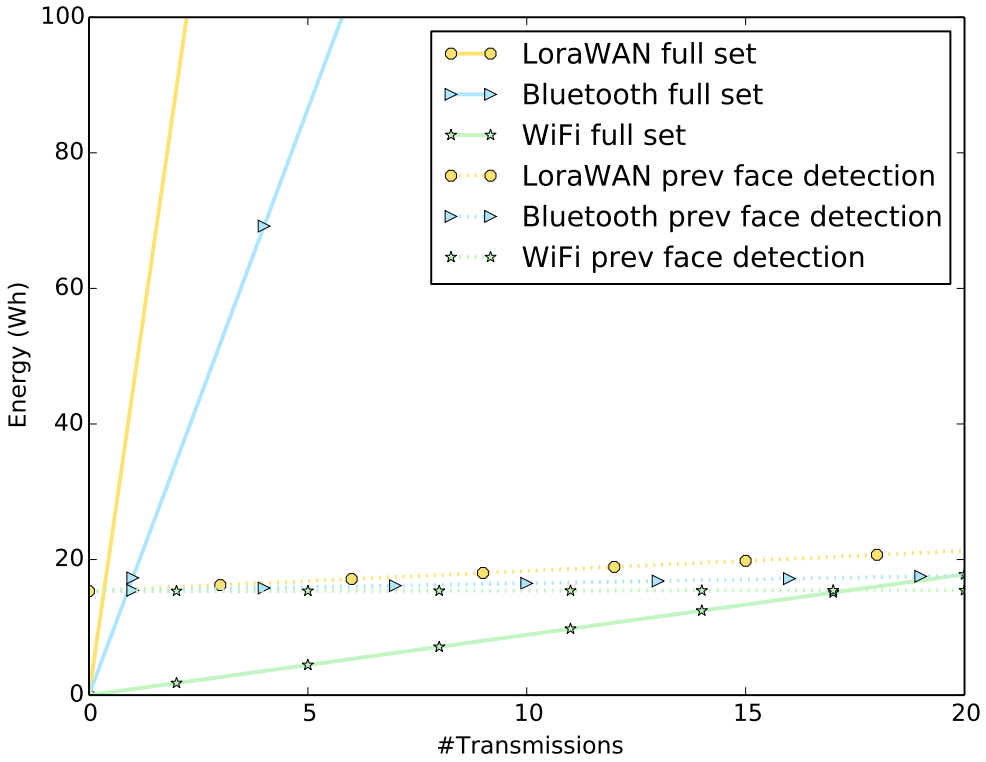


Figure 5.14: Energy consumed for a transmission with and without a previously applied face detection

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 LoRa interface (connected to a RPi), a Bluetooth interface (via two RPis) and a WiFi interface (on the RPi during a transmission

Name	$avg(t_c)$	$max(t_c)$	$avg(t_i)$	$\sum$ bytes	E_{trans}	E_{comp}	E_{sum}
Disaster ONFs	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 ONFs	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 ONF	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 ONF	10.20 s	27.81 s	2.08 s	131.3 MB	180.26 Ws	836.2 Ws	1016.48 Ws
Star-share ONF (lightw.)	11.01 s	31.04 s	4.08 s	354.8 MB	487.10 Ws	21.0 Ws	508.10 Ws
Star-share ONF (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

Table 5.4: Scenario tests

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 5.14 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 these links a face detection before the first transmission is useful, the break-even point for WiFi 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.

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 WiFi access point and ad-hoc characteristics, traffic shaping, as well as simple network bridges for direct linking of certain nodes [Ahr+08].

Topologies

We investigated 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 functions like face detection. Figure 5.15 shows a screenshot of *Disaster* scenario, a topology as it could be found in an emergency event. In the disaster site, 22 people are trapped building a partial mesh and 15 rescuers are moving in the direction of the trapped people also building a partial mesh. When the rescuers approach the building the two mesh networks merge. To support the approaching rescuers, the trapped people document different aspects of the disaster using their mobile phones. A LoRa link is established to enable basic text communication and to propagate the interests of the rescuers.

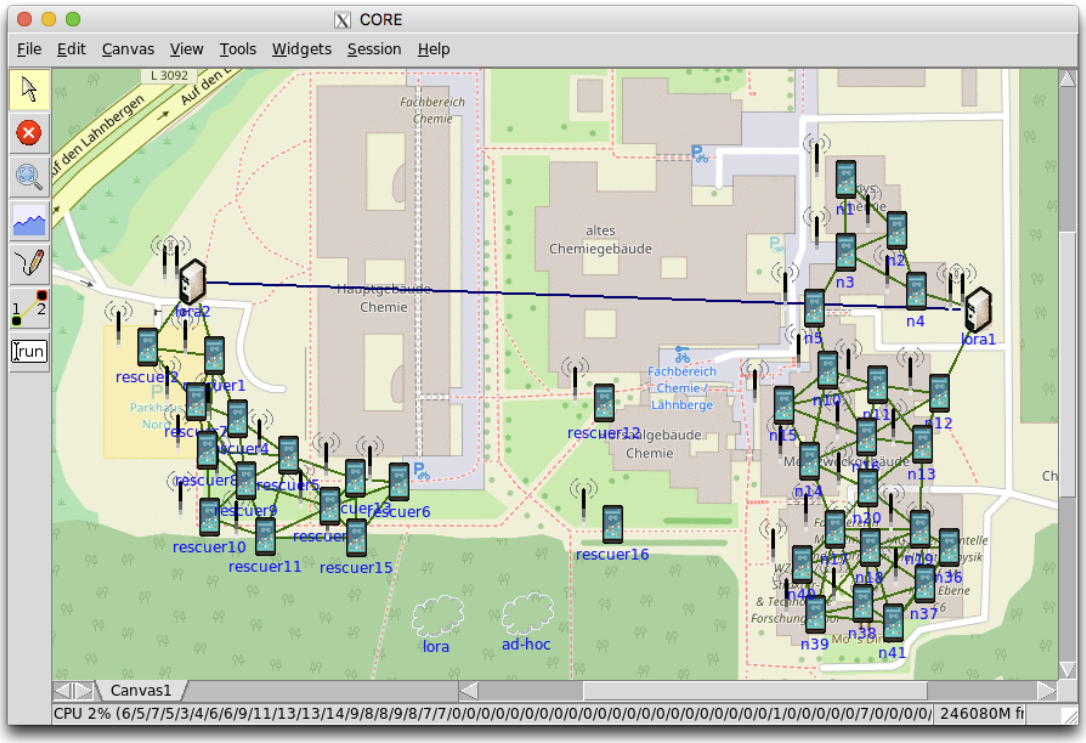


Figure 5.15: Disaster scenario modeled in CORE. Rescuers approaching from the left, trapped people on the right.

Simulation Results

Based on these topologies, we performed several simulation experiments. The results are presented in Table 5.4. 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 other nodes act as content consumers. The *Star* scenarios are further evaluated in two different configurations. In the lightweight 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 investigate different metrics applicable to the proposed mechanism (e.g., interest propagation 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 similar to our scenario in Section 5.2.4: 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 5.4. 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 are listed. The latter is modeled by (i) E_{trans}, the energy consumed during transferring data at 10 Mbps via WiFi (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 5.4 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 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 to 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 also cannot utilize ONFs for better energy efficiency. In the first two *Star-share* scenarios, content transmission times were reduced by a small amount, showing again a small advantage of the 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.

5.2.7 Summary

In this section, opportunistic named functions as a novel approach to operate ICN-DTNs during emergencies, have been presented. Affected people and first responders use their mobile devices to 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 as well as on locally optimal decisions using criteria such as battery lifetimes and device capabilities. In the presented emergency scenario, they were used to preprocess, analyze, integrate and transfer information extracted from images produced by their smartphone cameras, with the aim of supporting the search for missing persons and the assessment of critical conditions in a disaster area. Experimental results have shown that opportunistic named functions reduce network congestion and improve battery lifetimes in a heterogeneous network of battery-powered sensors, mobile devices, and mobile routers, while delivering crucial information to carry out situation analysis in a disaster. Using ONFs in our disaster scenario saves a significant amount of traffic (factor 4) and energy (36%) compared to not using ONFs by flooding the network with raw images.

Based on the results shown, this novel approach to operate ICN-DTNs is a smart solution in the sense of this thesis. For the comparison between information analysis cost and achievable quality, as shown in Figure 5.1 on page 80, the data of the experimental evaluation can be used. The comparisons are carried out within different scenarios, so the results also vary depending on the scenario evaluated. However, compared to a conventional DTN, in which the data is first transmitted and then a function execution is performed on the receiver node, there are some major advantages. The transmitted data alone is significantly reduced to up to 17.4% of the conventionally required total data transfer. Energy consumption has also been reduced to 32.7% in some scenarios, including the energy required for computations performed on the network devices. Particularly in scenarios with network congestion, such as the disaster scenario described above, this leads to faster or more complete delivery of data such as images from the scene of the accident, and thus to an improved QoS for the network.

5.3 Offloading Computational Workflows in Opportunistic Networks

5.3.1 Introduction

Opportunistic networking is useful for communication in scenarios where no infrastructure is available, if network connectivity is intermittent or error-prone. This is achieved using a store, carry and forward approach to transmit bundles hop-to-hop, from source to destination. Opportunistic networking can help first responders and victims in disasters, inhabitants in rural areas, and researchers in environmental monitoring of natural habitats to exchange data without relying on a working communications infrastructure [Gar11; Bau+18; Con+10]. Since mobile devices used in such scenarios typically have limited computational power, storage, or energy, offloading computational tasks can reduce load on initiating devices or even enable task execution, e.g., if specialized hardware is required [Kum+13]. Face detection in disaster scenarios could help first responders to save resources for essential communication [Con+10], and environmental monitoring with mobile sensor nodes is a current research topic [Bau+18].

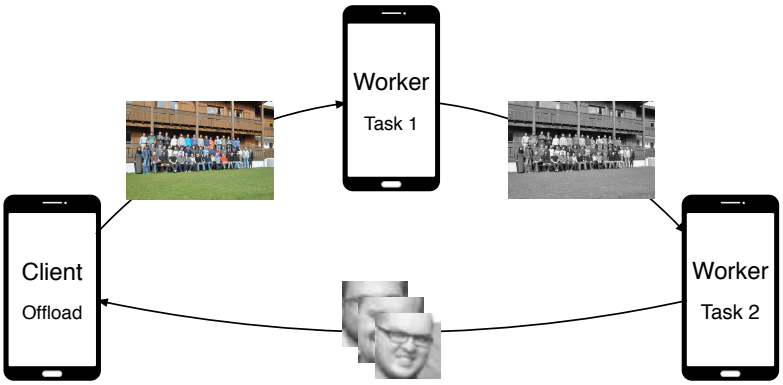


Figure 5.16: Illustrative example: executing a workflow on two workers.

In this section OPPLOAD is presented, a novel framework for offloading computational workflows in opportunistic networks. Fig. 5.16 shows an example for offloading a simple face detection workflow where Task 1 converts an image to grayscale, and Task 2 extracts faces and returns them to the client [Lam+17]. Using OPPLOAD, *clients* can assign the individual tasks of a workflow to particular remote execution platforms (*workers*) ahead of time or can leave the assignment open, i.e., each worker will search for another suitable worker just in time for the next task using our novel worker assignment algorithm, while both modes can be mixed in a workflow. Workers publish their capabilities and resources (available memory, remaining battery capacity, etc.), i.e., clients will only select capable workers. Furthermore, workflow tasks can be executed on multiple workers that are automatically selected to balance the overall load, based on a folded standard normal distribution and an innovative worker ranking system, where workers are rated based on their available resources, their capabilities, and the proximity to the calling client/worker, resulting in a novel approach ensuring that load on workers is distributed fairly in the network. No permanent connection to the worker nodes is needed. Depending on node mobility, results can still be delivered even after longer periods of isolation due to a disruption-tolerant networking (DTN) underlay.

To show the feasibility of our approach, we emulated up to 30 highly mobile nodes in different experimental settings, showing that the success rate of offloading increases by up to 40% with negligible overhead. Our Python implementation[22] and all artifacts[23][24] of this section are publicly available.

Parts of this section have been published in Artur Sterz, Lars Baumgärtner, Jonas Höchst, Patrick Lampe, and Bernd Freisleben. "OPPLOAD: Offloading Computational Workflows in Opportunistic Networks." in: *2019 IEEE 44th Conference on Local Computer Networks (LCN 2019)*. Osnabrück, Germany, Oct. 2019. DOI: 10.1109/LCN44214.2019.8990775.

5.3.2 Related Work

Workflow-based Approaches

To offload tasks to other mobile devices, Serendipity splits each task into smaller tasks that are either offloaded or not if no worker is found [Shi+12]. In a mobile cloud computing scenario, Ahn et al. [Ahn+18] start the execution of tasks locally and offload them to suitable cloudlets. Ravi and Peddoju [RP18] present an offloading algorithm where an application is partitioned into clusters containing tasks to decide whether to offload, based on a density-based clustering algorithm.

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

Proximity-based, Movement-based, and Social Approaches

COMET [Gor+12] is a framework for offloading parts of applications to neighboring nodes to speed up their execution. Mtibaa et al. [Mti+13] propose a framework where a task is offloaded to mobile devices that belong to the same social context, e.g., the same household or a group of first responders in a disaster scenario.

Wang et al. [WLJ14] present an offloading scheme for opportunistic networks where mobility patterns are analyzed to estimate the number and duration of contacts for the offloading decision. Zhang et al. [ZNW15] consider the load of a device, the availability of cloudlets, and user mobility to maximize the probability of successfully offloading tasks. Honeybee [FLR16] includes a work sharing algorithm that employs nearby nodes to execute tasks based on job stealing.

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

[22] https://github.com/umr-ds/OPPLOAD
[23] https://github.com/umr-ds/OPPLOAD-experiments
[24] https://ds.mathematik.uni-marburg.de/oppload/oppload_results.tar.gz

Offloading in Cloud Environments

Deng et al. [Den+15] decide for each task of a workflow whether it should be offloaded to the cloud or executed locally, based on the capabilities and the movement of nodes. Chatzopoulos [Cha+16] use an incentive mechanism where users have to define how many resources they are willing to spend for executing offloaded tasks. Chowdhury et al.[Cho+18] migrate tasks between cloud, mobile devices, or robots by considering energy, latency, and task execution deadlines.

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

Mobile Cloud, Edge, and Fog Environments

Fan et al. [Fan+18] present an approach where a base station in a mobile cloud scenario can either execute an offloaded task itself or further offload it to another base station. Using a fuzzy decision engine, Flores et al. [FS13] consider multiple criteria like CPU power to decide whether a task should be offloaded to a mobile cloud server. Yang et al. [Yan+13] offload computations in mobile cloud scenarios to maximize the throughput of applications. Chen et al. [Che+16b] formulate a game theoretic approach for offloading tasks in a mobile cloud scenario. Bellavista et al. [BZS17] present a computation offloading approach, where tasks are offloaded to mobile edge cloud instances and the results are return over the same node or a different one, if the user has moved in the meantime. Zhang et al. [Zha+18] introduce a task allocation scheme where social sensing applications are offloaded to edge servers to maximize a node's payoff by saving energy. Yang et al. [Yan+18] propose an algorithm to offload tasks to a nearby edge server.

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

Other Approaches

Funai et al. [FTH16] present an approach that minimizes energy consumption by offloading computations across multiple hops in an ad-hoc network. Zanni et al. [Zan+17] propose an approach to split arbitrary Android apps into smaller tasks that can be offloaded. Sterz et al. [Ste+17] present a framework for remote procedure calls in disruption-tolerant networks with separated control and data channels to cope with short contact durations. Internet-of-Things devices use more capable devices that are reachable within one hop to execute a task [ES18]. Feng et al. [Fen+18] present an approach where mobile devices offload tasks to other mobile devices via cellular base stations without prior knowledge of the devices' resources.

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

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

5.3.3 OPPLOAD's Design

Workflow-based Computations

OPPLOAD supports *workflow-based computations* where a client defines a workflow that consists of a chain of tasks. The client assigns each task to a worker, and OPPLOAD will take care of the execution order, even in unpredictable network situations. Furthermore, OPPLOAD transparently passes inputs and outputs between the different tasks of a workflow. Connectivity is achieved using protocols for disruption-tolerant networking (DTN), while we assume that the communication overhead in terms of CPU and memory resources for remote execution is negligible [Bau+16].

Worker Addressing

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

Ahead of Time

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

Just in Time

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

If a client does not assign a task to a specific worker, OPPLOAD will transparently chose a suitable worker by passing the workflow description through a number of steps that are part of worker assignment, as shown in Fig. 5.17. During the first step of worker assignment, a worker with an offer for executing a desired task will be searched in the local database. If the search is successful, the task will be executed on this worker. This mode is helpful when it is not clear whether a worker is available for a task.

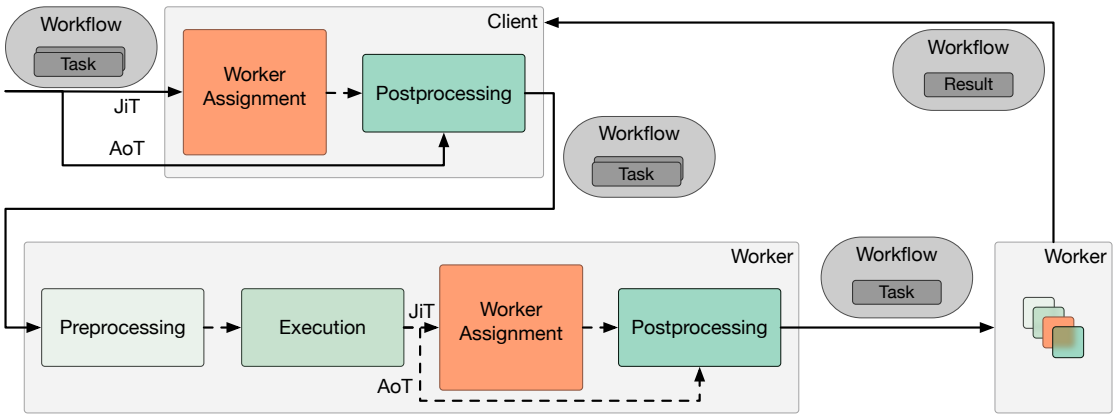


Figure 5.17: Architecture of OPPLOAD client and worker showing a possible workflow with Ahead of Time (AoT) or Just in Time (JiT) worker assignment.

Worker Capabilities

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

Worker Assignment

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

Error Handling

Bundle delivery in opportunistic networks cannot be guaranteed. If a worker disappears in OPPLOAD before it could execute an assigned task, the client would wait infinitely long. Therefore, users can specify a time-to-live (TTL) for a workflow. This has two implications. First, the client stops waiting for the results after the TTL has expired, making it possible to re-issue the workflow. Second, a worker will not execute a task if the TTL is expired, which preserves resources on workers. This ensures a defined behavior in cases where no result can be retrieved in time.

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

5.3.4 Implementation

We implemented OPPLOAD based on the bundle store implementation, *Rhizome*, of the Serval Mesh [Gar11], which uses a simple epidemic DTN routing protocol. OPPLOAD is written in Python and uses Rhizome's RESTful API for handling all network-related duties. In previous work, we have conducted an in-depth evaluation of Serval in various experiments with different network setups and usage patterns [Bau+16].

Offering a Service

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

Executing a Workflow

To execute a workflow, a user splits it into tasks to be executed across multiple workers. All tasks have to be described in a workflow description containing the desired worker (either AoT or JiT), the name of the task, and all required parameters, for each task. A workflow description must include at least one task. This workflow description has to be provided to the OPPLOAD client that handles the remaining parts transparently.

A workflow description has the following form. First, a task has to be assigned to a worker, which can be an address for AoT mode or a placeholder indicating that JiT mode should be used. Then, the name of the service to be executed has to be given, followed by all parameters. Using another placeholder indicates that the output of a task should be the input for the next task. Each task can only have one result, and the placeholder is allowed only once per task. Finally, a task can have requirements that are only used during assignment for this particular task. After specifying the workflow, OPPLOAD will assign a worker to the first task, if applicable. The first step is to rank all workers, which is based on the requirements, as introduced in Section 5.3.3. For each metric, a weighted rank is calculated and summed up, using the weight and the requirements as well as the worker's capability for the particular metric. Workers are sorted based on their ranking, and a random worker is selected based on the folded normal distribution with location parameter $\mu = 0$ and scale parameter $\sigma = 1$. All files required for a task, the workflow description itself, and task results or errors will be packed into an archive that will be sent as an encrypted bundle to the selected worker. By packing everything in a single archive, fragmentation in transmission is avoided, and a worker is guaranteed to have everything required for processing the task.

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

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

5.3.5 Experimental Evaluation

Test Setup

To evaluate OPPLOAD in a realistic manner, the network emulation framework *Common Open Research Emulator* (CORE) was used. In contrast to simulation approaches like NS-3, CORE uses Linux namespaces to execute binaries and scripts natively, which gives us the opportunity

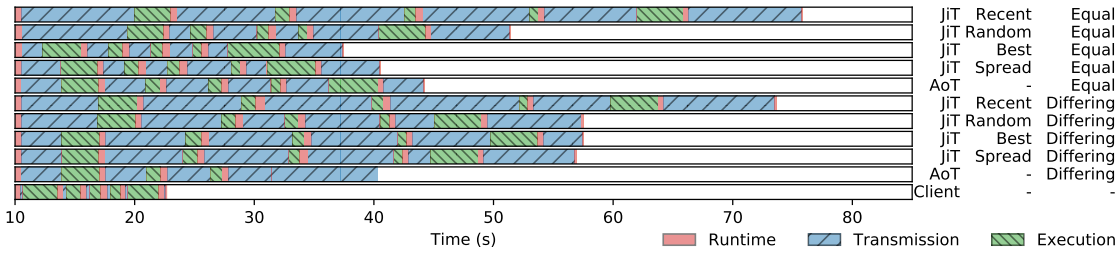


Figure 5.18: Exemplary overall workflow time in different configurations.

to evaluate software and frameworks as close to reality as possible by still being able to scale the experiments easily [Ahr+08].

Test Cases

To evaluate OPPLOAD, the algorithm of Lampe et al. [Lam+17] for detecting faces in images on smartphones was adapted. The workflow of this algorithm has five tasks. The first task is to denoise an image. The second task is to scale the image up by 10% to increase the probability of fitting a possible face into the detection window. The third task is to crop the image by 10% to decrease the image size, which speeds up the detection time while maintaining a high detection accuracy. The fourth task converts the colored image into an 8-bit grayscale image, which additionally speeds up face detection while maintaining the same detection accuracy. The fifth task detects faces on the preprocessed image. These five tasks are executed on five different workers in the network. In every experiment, the bandwidth of the network links was set to 54 Mbit/s, and a delay of 20 ms was used. All nodes were configured as workers. In JiT mode, we compared four worker assignment algorithms: (i: *recent*) selecting the worker whose offer arrived most recently, (ii: *random*) selecting a worker randomly, (iii: *best*) selecting the best available worker based on our rating, and (iv: *spread*) using the algorithm described in Section 5.3.3 to spread the load among the best available workers. Since OPPLOAD is designed for networks with mobile devices as workers, the weights for the worker rating were set to keep the tasks on nodes with high energy reserves and spatially close to the client. Therefore, available energy and distance were weighted with 30%, CPU with 20%, and available memory and free disk space with 10%. We modelled energy using a virtual energy unit e. These were used to model energy consumption for each task related to the task's execution time, meaning that the longer a task takes on average, the more e is consumed. In our experiments, a service offer from a worker was set to expire after 120 seconds, as described in Section 5.3.3. Finally, workers announced their capabilities every 2 seconds, since this is the sweet spot announcement interval, as shown by Baumgärtner et al. [Bau+17].

Baseline Evaluation

The baseline tests show how OPPLOAD performs under good network conditions. For these tests, twelve static nodes were arranged in a ring, where each node had exactly two neighbors and only the first node was a client. In AoT mode, workers were selected at the start of an experiment in the same order as they appear in the network, always skipping one node. The

same workers were used for all AoT experiments for comparability. To evaluate the effect of worker capabilities, this setup was first executed with all workers equally capable of executing a task and a second time where we used the following capability distribution: 20% (2) of the workers were capable with no constraints, 40% (5) were also capable, but had less energy reserves, 30% (3) could execute the task, but with limited capabilities (like little available memory) and 10% (2) were not capable to execute the task at all. The capabilities were modeled using available disk space, memory, CPU resources, and energy e, which was reduced according to the above description. Since worker assignment requires randomness, our random number generator was initialized with 25 different seeds. Finally, we executed the experiments also on the client to have a benchmark for comparison.

Workflow Profiling

To analyze the overhead of OPPLOAD, workflow processing was split into three phases: (i) runtime (red) of the OPPLOAD implementation, i.e., pre- and postprocessing and worker assignment in JiT tests, (ii) transmission time (blue) for transmitting the bundle, and (iii) execution time (green) of the task itself. The colors refer to Fig. 5.18. The x-axis shows the workflow execution time, each bar denotes a specific configuration.

As shown in Fig. 5.18, OPPLOAD does not introduce significant processing overhead. The workflows are offloaded from the clients 10 seconds after the start of the experiment. Regardless of the test configuration, postprocessing and worker assignment require about 1 second, while preprocessing can be neglected. The execution time depends on the task. While scaling, cropping, and grayscaling only require about 2 seconds, denoising and detecting faces can take up to 6 seconds.

If AoT addressing is used in known topologies, users can estimate a workflow time range in which it finishes. The downside is that if a worker is not capable of executing a task, the entire workflow will be stopped, as indicated by the second last bar in Fig. 5.18. Therefore, tasks should only be explicitly assigned in cases where no other option is desirable, or if a task must be handled by a specific worker.

The major overhead is introduced by transmitting the bundles across the network. The last bar of Fig. 5.18 shows the same workflow executed on the client, thus no networking is needed. The entire workflow needs about the same time as two to three tasks in the JiT tests, depending on the worker assignment. Although overhead is introduced by network related operations, it can still be better to offload workflows than executing them locally. First, the client may not be able to execute the tasks due to resource constraints or other limitations. Second, the longer the tasks take to be executed, the more negligible the communication overhead becomes. Finally, the decision whether to offload or not also depends on the number of hops between the offloading node and the worker, as indicated by Graubner et al. [Gra+18a]. For OPPLOAD, we assume that the user decides whether to offload during the creation of the workflow.

Tables 5.5 and 5.6 show the average time needed for the parts of a workflow (the numbers in brackets show the standard deviation) in seconds. Table 5.6 indicates that the overall workflow time highly depends on the worker assignment in the JiT experiments. The recent worker assignment with an average of about 64.48 seconds requires the longest time, due to the

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

Table 5.5: Average runtimes of workflow parts in the ring scenario in client-only tests and using AoT addressing.

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

Table 5.6: Average runtimes of workflow parts in the ring scenario using JiT addressing and all four assignments.

long distance between the nodes, since their offers take longer to reach the client and thus arrive more recently. The standard deviation is also relatively high with more than 10 seconds, indicating long running tasks and differing results. The random worker assignment achieves better results with about 46.35 seconds on average and a deviation of 4.25 seconds, since closer workers are chosen. Always selecting the best available worker leads to significantly lower workflow times, requiring about 37.49 seconds, but with a standard deviation of 9.99 seconds. Finally, using the spread assignment algorithm, the workflow time does not significantly differ from the previous assignment algorithm, using about 37.94 seconds, but has a better standard deviation of 7.11 seconds. If all workers are equally capable, the workflow times using the best worker or the spread algorithm do not differ. This shows clearly that in terms of workflow time, the algorithm using the best workers and our spread approach outperform the other approaches. But since not all workers are equally capable in the different capability tests, tests using the best worker have a broader standard deviation, since the capable workers are further away in the topology. This means that always using the best worker is slightly faster than using the spread algorithm, but is more unpredictable in how long the execution of a workflow will take, since the very best workers will be worn up and worse workers have to be chosen consequently. Therefore, we propose our spread algorithm as the best available solution. Executing a workflow locally at the client would only require execution time and runtime, since the networking part is not needed. As shown in Table 5.5, the total execution time is about 12.52 seconds and is pretty stable with only about 300 ms deviation. Finally, the AoT mode needs about 34.15 seconds in total and is also stable with only about 400 ms deviation. Since in AoT mode a worker is always two hops away from the next hop, the transmission is even faster than using JiT mode with the best worker assignment.

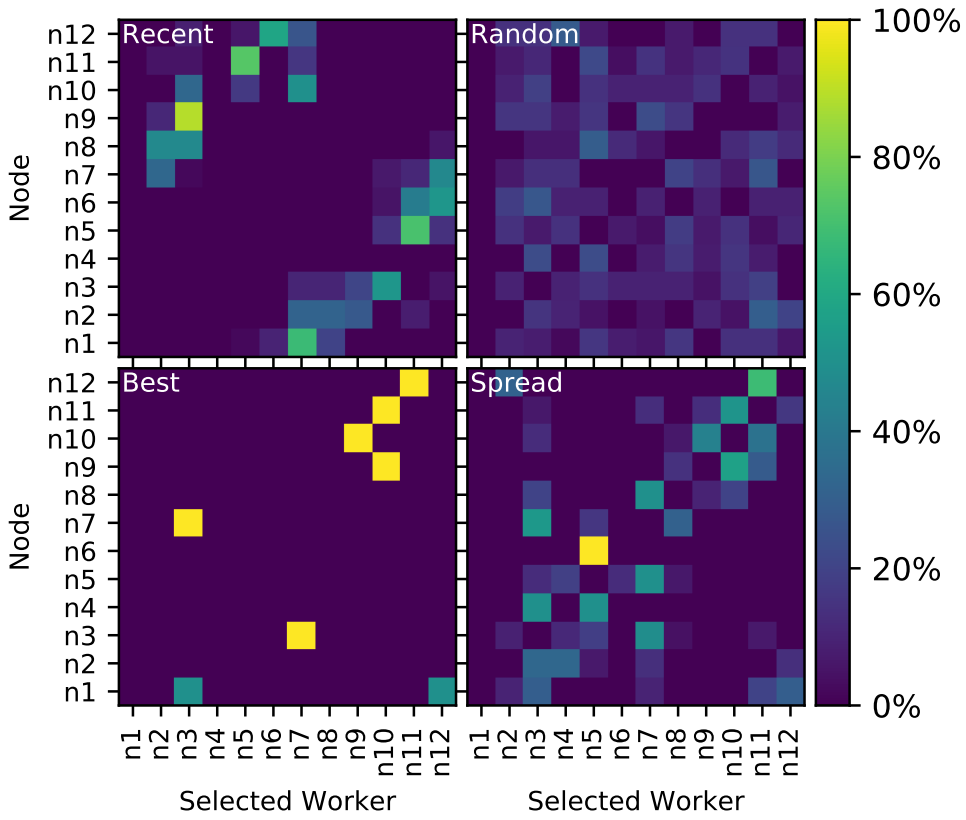


Figure 5.19: Worker selection in the ring topology just in time assignment scenarios.

Worker Load Distribution

Fig. 5.19 shows the worker load distribution in all four worker assignment algorithms using JiT mode. On the y-axis, the calling nodes are shown, whereas on the x-axis the assigned worker is denoted. The lighter the color, the more often a particular client selected a particular worker.

The recent selection approach spreads the load over particular nodes, but almost always selects a worker on the opposite side of the network, leading to long-running workflows. Using a random worker, the workload is distributed on nearly all available workers. Although this leads to a fair load distribution, the profiling analysis shows that this approach does not necessarily give the fastest workflow execution times. Additionally, tasks are sent to spatially far away workers, leading to the same problems of long transmission times and network splits in mobile networks as in the recent approach. Always using the best available worker keeps the workflow execution spatially close, and the overall runtimes are the lowest achievable, but with an unfair load distribution, which disadvantages close and powerful workers over others that are also able to execute a task. In dense networks with a high offloading frequency, this could lead to overloaded nodes and empty batteries, which in the end would be less beneficial for the overall performance. Finally, our approach spreading the load on the best workers leads to the best overall results. Close and powerful workers are preferred over others, while less powerful workers also have chances to be selected. Overall, as previously shown in Table 5.6,

the workflow times are nearly as good as always selecting the best worker. Thus, our algorithm should be used instead of the other presented approaches.

CPU, Memory, and Bandwidth Utilization

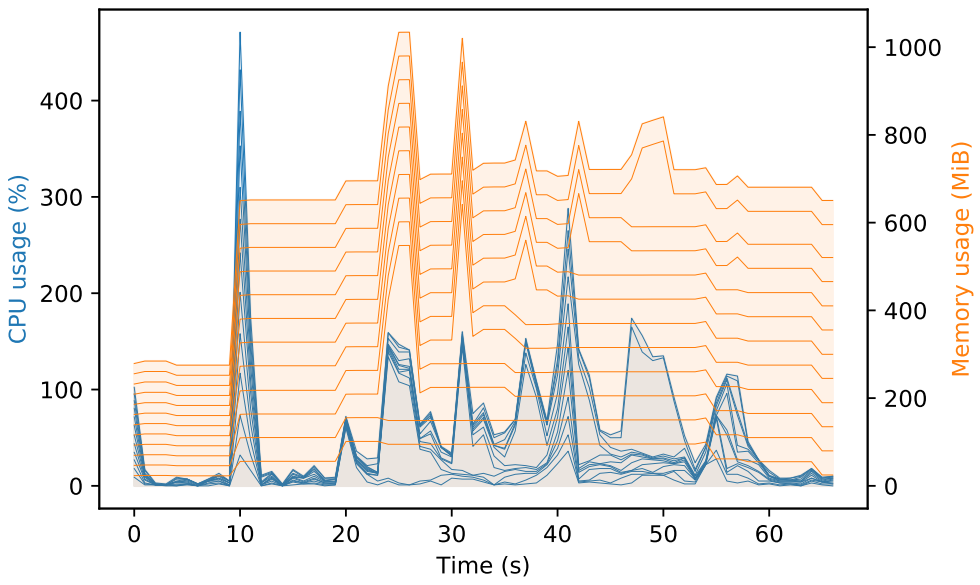


Figure 5.20: CPU and memory utilization in AoT mode; every worker capable.

Fig. 5.20 shows the CPU and memory utilization of an experiment in AoT mode where every worker was equally capable to execute a task. On the x-axis, the time is shown, whereas the left (blue) y-axis denotes the CPU usage and the right (orange) y-axis shows the memory allocation. In both graphs, the resource usages of all nodes are stacked, whereas 100% CPU load means that one CPU core of the emulation host is fully utilized (the emulation host had 80 CPU cores and 256 GB RAM, both are not exceeded).

During the first 10 seconds, the test is set up (the emulated nodes are started, configuration files are prepared, etc.). After 10 seconds, Serval and OPPLOAD are started, which require many computations (e.g., loading Python interpreters into memory, computing hashes for the worker capabilities), and the CPU utilization has a high peak with more than 400% CPU. During the experiment, five peaks can be identified, which are the five tasks of the workflow. The CPU peaks are more blurred, since not only during the task the CPU is used heavily, but also during transmitting the result to the next worker using Serval. Memory usage shows that on average every node requires about 60 MB of memory, while the execution of a task leads to peaks, due to the fact that the image and the task binary itself have to be loaded into memory.

OPPLOAD in Action

In the final set of experiments, we studied a 30 node network using five different random-waypoint mobility models, since randomly moving nodes is the most challenging scenario

in opportunistic networks. The worker capabilities were set differently in all experiments, as defined in Section 5.3.5. Furthermore, we evaluated the behavior with 5 and 10 clients that offload tasks at the same time in the network at the start of an experiment, which can lead to workers executing multiple tasks simultaneously. To simulate an IEEE 802.11g network, which is still widely used especially in decolping countries, with a bandwidth of 54 Mbit/s, a basic range model for the Wi-Fi nodes with 40 meters of range was used. The mobility model was configured for 30 nodes, walking randomly in an area of about 1.7 km^2 at a speed between 0.8 m/s and 1.9 m/s or rest for up to 60 seconds, which corresponds to human walking speed. This setup leads to relatively small mesh networks that are appearing and disappearing during the execution of the experiment. Overall, 200 experiments were executed, all using JiT mode. An experiment finished either successfully, meaning that all clients received their results, or it was stopped after 30 minutes.

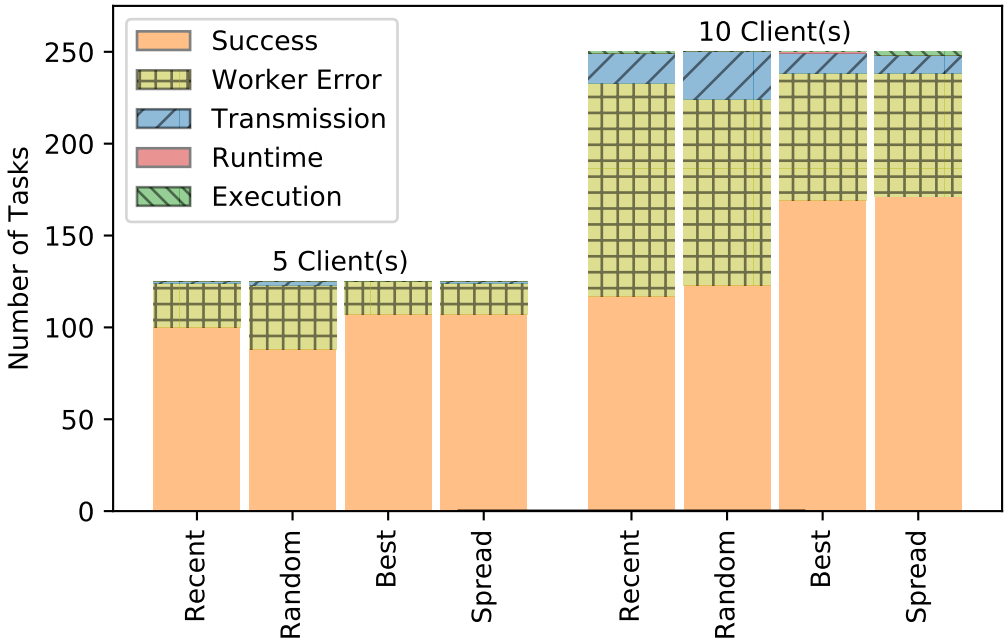


Figure 5.21: Final workflow states, by number of active clients in JiT mode.

Fig. 5.21 shows the final states of the workflows executed in the specific scenarios, where the bars are grouped by the number of clients per experiment and worker assignment. The y-axis shows the number of tasks in a particular state. The first case is a successful workflow (*Success*), where a workflow was offloaded, all tasks could be executed, and the result arrived at the client. Second, OPPLOAD performed as intended but errors occurred as discussed in Section 5.3.3 (*Worker Error*) and the client could successfully be informed about this error. An experiment stopped in the *Transmission* state, if a task was transmitted to the next worker, but not received until the end of the experiment, e.g., if the recipient cannot be reached due to network fragmentation. Due to experiment abortion while OPPLOAD was in a runtime state or a worker executed the task itself, it is denoted as *Runtime* and *Execution*, respectively.

Experiments using the recent assignment mode have the lowest success rates, which is due to the fact that workers are selected that are far away and the offers arrive late. Using a random worker increases the number of successes slightly. Using the best worker available,

Assign.	Exec. (s)	Runt. (s)	Transm. (s)	Total (s)
Recent	8.7 (0.64)	5.0 (1.89)	269.0 (336.37)	282.8 (338.91)
Random	8.9 (1.02)	5.0 (1.84)	254.9 (300.75)	268.8 (303.61)
Best	8.9 (0.62)	5.2 (1.80)	135.5 (191.26)	149.6 (193.68)
Spread	8.9 (0.68)	5.1 (1.95)	234.2 (300.75)	248.1 (303.61)

Table 5.7: Average runtimes of tasks in mobile JiT scenarios in seconds.

all tests were either successful or the client was informed about an error when 5 clients are used. Our spreading approach is as good as using the best worker in terms of successful workflows or errors returned in time. The fact that even using the best worker does not lead to 100% successful executions is due to the worker capabilities and the transmission time in opportunistic networks. A worker updates its capabilities after executing a task, which can lead to the situation that another task is offloaded to the worker, even though it is not capable anymore. The falsely assigned worker will decline task execution and inform the client.

Table 5.7 shows the average workflow runtimes over all mobile experiments. It is evident that using our spread algorithm gives better results than random assignment and using a recent worker. Note that the transmission times (and thus also the total times) have a rather high standard deviation. This is due to the mobility of the nodes and potentially disappearing links between two nodes, resulting in re-transmissions. These increase the time, whereas many transmissions are successful within the first try, reducing the mean transmission time.

To summarize, OPPLOAD introduces negligible overhead in terms of CPU load or memory consumption and supports efficient offloading of computational workflows on resource-constrained devices in opportunistic networking scenarios.

5.3.6 Summary

We presented OPPLOAD, a novel framework for offloading computational workflows in opportunistic networks, with two addressing modes, workers publishing their capabilities and available resources, a worker assignment algorithm, appropriate error handling, and network cleanup to reduce network load. Experiments with up to 30 emulated mobile nodes showed that worker assignment is important for speeding up workflow execution and for spreading the load fairly on spatially close but powerful workers, which increases the rate of successful offloadings significantly.

OPPLOAD is a smart solution in the sense of this thesis, as the presented results have shown. A classification based on information analysis cost and achievable quality, as presented in Figure 5.1 on page 80 cannot be made directly, since a new system has been created for which there is no conventional alternative. Nevertheless, a similar functionality could be realized with the help of remote procedure calls. However, since in opportunistic networks it is especially likely that connections will be interrupted, such a system design is only of limited help in achieving the functionality. In addition, the composition of workflows from individual tasks also ensures

that the effort can be distributed among different participants. We argue that by avoiding the repetition of computations, in particular due to disconnected connections and participants that have left the network, the information analysis cost is lower than for a comparable system that is based on conventional remote procedure calls. At the same time, the QoS is higher, since error handling can already take place within a workflow, and individual tasks can be dynamically executed on other workers. The decomposition and distribution of workflows, coupled with local decision-making in the network, therefore leads to a higher QoS.

5.4 DTN7: An Open-Source Disruption-tolerant Networking Implementation of Bundle Protocol 7

5.4.1 Introduction

Delay- or disruption-tolerant networking (DTN) is useful in situations where a reliable connection to a communication infrastructure cannot be established, e.g., during environmental monitoring in remote areas, if telecommunication networks are destroyed as a result of natural or man-made disasters, or if access is blocked due to political censorship. In DTN, messages are transmitted hop-to-hop from network node to network node in a store-carry-forward manner. There might be larger time windows between two transmissions, and the next node to carry a message might be reached opportunistically or through scheduled contacts.

There are several mobile DTN appications, such as FireChat [Ope19] and Serval [Gar11], that rely on peer-to-peer networks of smartphones, where the pre-installed Wi-Fi or Bluetooth hardware of the mobile devices is used to create a large mesh network. µPCN [FW15] is a special purpose DTN application for planetary communication, and IBR-DTN [Doe+08] is a popular DTN platform, but does not implement the recently released Bundle Protocol (BP) Version 7 [BFB22].

In this section DTN7 is presented, which is the first and only freely available, open source implementation of the most recent draft of Bundle Protocol Version 7 (BP7). DTN7 is designed to offer extensibility by allowing developers to easily replace or add individual components. DTN7 is a general purpose DTN software with support for several use cases, such as enabling communication in disaster scenarios or providing connectivity in rural areas. Our contributions are:

- We provide a memory-safe and concurrent open-source implementation of BP7, written in the Go programming language.
- With its highly modular design and its focus on extensibility by providing interfaces to all important components, DTN7 is a flexible basis for DTN research and application development for a wide range of scenarios.
- We compare DTN7 with other well-known DTN systems including Serval, IBR-DTN, and Forban, using the CORE network emulation framework.
- Several experiments to mimic different DTN test cases, i.e., a chain of up to 64 nodes with different payload sizes, are conducted.

- The presented DTN7 software[25], the evaluation framework and its configurations[26], and the experimental fragments[27] are freely available.

Parts of this section have been published in Alvar Penning, Lars Baumgärtner, Jonas Höchst, Artur Sterz, Mira Mezini, and Bernd Freisleben. "DTN7: An Open-Source Disruption-tolerant Networking Implementation of Bundle Protocol 7." in: *18th International Conference on Ad Hoc Networks and Wireless (ADHOC-NOW 2019)*. Esch-sur-Alzette, Luxemburg, Oct. 2019. DOI: 10.1007/978-3-030-31831-4_14.

5.4.2 Related Work

This section briefly reviews relevant publications in the area of DTN software.

DTN Software Implementations

IBR-DTN [Doe+08] is a lightweight, modular DTN software for terrestrial use. The Interplanetary Overlay Network (ION) focuses on the aspects of extreme distances in space [Bur07]. DTN2 is the reference implementation of the BP, developed by the IETF DTN working group [Dem+03]. These three implementations are based on RFC 5050, i.e., BP Version 6 [SB07].

Designed for small satellites in low earth orbit, µPCN can be used to connect different regions of the world. It also implements BP Version 6, as well as an older draft of version 7 [FW15]. Furthermore, an older version of BP7 is implemented in Terra [Rig18].

Serval focuses on node mobility by providing implementations that run on smartphones, as well as by incorporating different radio link technologies [Gar11]. Forban is a peer-to-peer file sharing application that uses common Internet protocols like IP and HTTP to transmit files in a delay-tolerant manner [Dul16]. With FireChat [Ope19], it is possible to send messages via DTN without relying on Internet access or direct peer contacts.

Many of the mentioned DTN systems implement the BP as specified in RFC 5050 [SB07]. While some implement a draft of BP7, none of them implements the most recent draft. Serval, Forban, and FireChat have their own protocol definitions, which are not compatible with the BP. Furthermore, the mentioned implementations cannot be extended in a modular manner, are not written in developer-friendly high-level programming languages and are not intended as general purpose DTN platforms, but are designed for specific use cases. FireChat is not freely available, and thus cannot be extended.

[25] https://github.com/dtn7/dtn7-go
[26] https://github.com/dtn7/adhocnow2019-evaluation
[27] https://ds.mathematik.uni-marburg.de/dtn7/adhoc-now_2019.tar.gz

DTN Software Evaluations

IBR-DTN, DTN2, and ION were evaluated by Pöttner et. al [Pöt+11]. For a payload of 1 MB, DTN2 and IBR-DTN produced almost identical results. ION was slower in the conducted measurements. Furthermore, the interaction of the three DTN implementations was evaluated by transferring bundles between them, and the times measured varied significantly.

IBR-DTN was used to evaluate the connection between a stationary DTN node and a moving vehicle [Doe+08]. This vehicle passed the stationary node at an average speed of 20 km/h, and the transmission rate was measured in relation to the distance. Data could be transmitted within a range of about 200 meters.

Serval was experimentally evaluated in our previous work (cf. Section 5.1), for scenarios with 48 nodes in a hub topology, 64 nodes in a chain topology, and 100 nodes in disjoint islands connected over time. The results indicate that Serval can achieve high network loads, while CPU usage remains relatively low.

5.4.3 Bundle Protocol Version 7

This section gives an overview of bundle protocols, referring to RFC 4838 [Cer+07] and the current version 7 of the Bundle Protocol (BP) [BFB22].

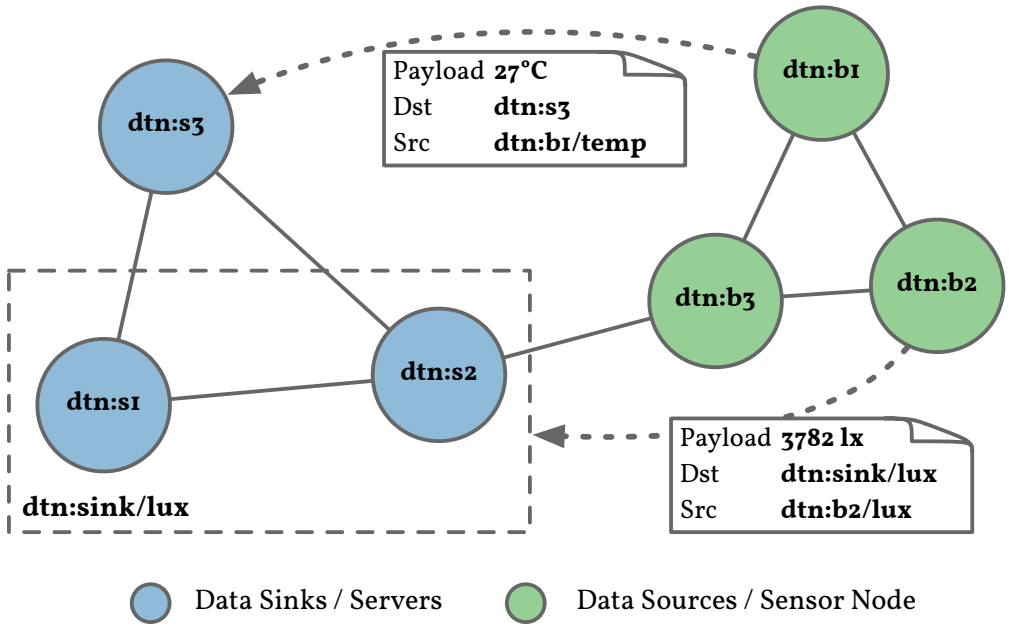


Figure 5.22: Example sensor node scenario with multiple endpoints.

Basic Concepts

Endpoints.

In DTN, there are nodes and endpoints. Nodes exchange bundles according to the store-carry-forward principle. Bundles are addressed at endpoints, or more precisely, their characterizing *Endpoint Identifier* (EID), which might not be a currently existing part of the network. Fig. 5.22 shows an example of a scenario, where sensor nodes produce readings to be consumed by data sinks. The temperature bundle is addressed directly to `dtn:s3`, where the lux bundle is headed to `dtn:sink/lux`, an EID that is handled by two nodes, and thus a multicast. BP7 is endpoint scheme agnostic and supports the null endpoint for anonymous bundles. In BP version 6, only endpoints are defined, so it is not possible to address dedicated nodes.

Bundles and Blocks.

Packets in a DTN consist of multiple *Blocks* to form logical units called *Bundles*. In Fig. 5.23, an example bundle containing the mandatory Primary Block, and two Canonical Blocks, namely a Hop Count Block and the actual Payload Block, is shown, following the example of Fig. 5.22.

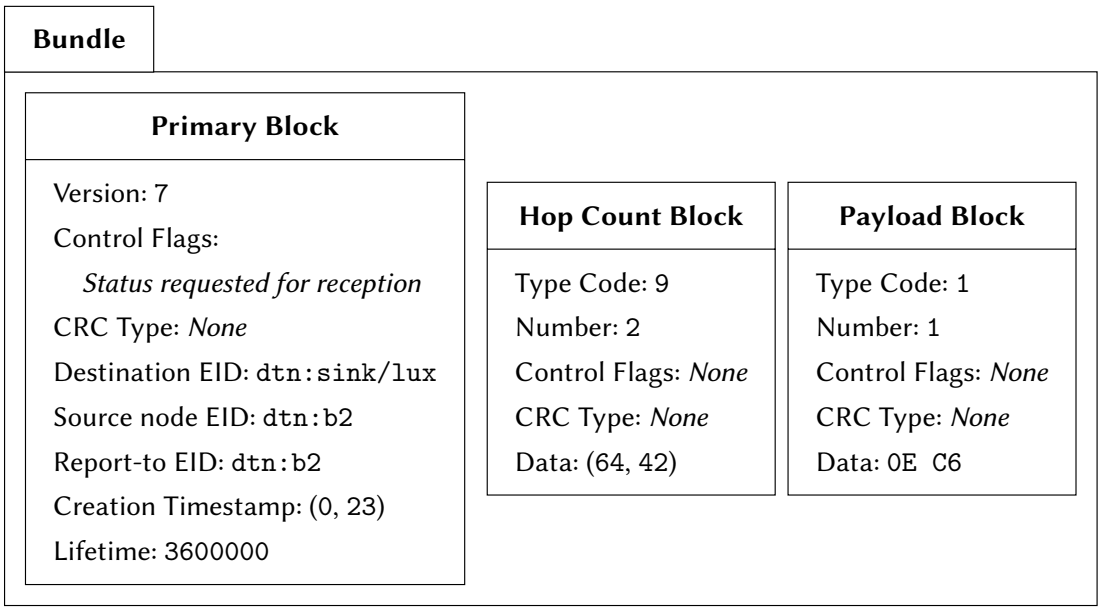


Figure 5.23: A bundle transmitting a lux value from `dtn:b2` to `dtn:sink/lux`.

Primary Block. Each bundle begins with a (since BP7 immutable) *Primary Block* (see Fig. 5.23), containing meta-information about the bundle with the following fields: Version; Bundle Processing Control Flags to provide information on the bundle, including fragmenting and reporting information; an optional CRC Checksum (added in BP7 and not available in BP version 6); Destination EID, Source Node ID and Report-To EID, as endpoints for administrative records regarding this bundle; Creation Timestamp, consisting of the actual timestamp and an

incrementing sequence number; Maximum Lifetime of a bundle, expressed in microseconds after creation time; Fragment Offset and Total Data Length, if fragmented and indicated by the bundle process control flags.

Canonical Block. Payload and Extension Blocks in Fig. 5.23 are summarized as *Canonical Blocks*. These contain a payload in addition to a few block-specific characteristics. A Canonical Block consists of a Type Code to identify the kind of block, Number to address the specific block, Control Flags and Data.

The actual payload of the bundle is located in the Payload Block at the end of each bundle. In addition to sending user data from application programs, status information is also sent within bundles, called Administrative Records, automatically created and sent by DTN software as a response to a previous bundle. Extension Blocks are Canonical Blocks containing further information relevant for a DTN router depending on its configuration. In contrast to BP version 6, the BP7 specification defines the Previous Node Block, Bundle Age Block, and Hop Count Block, and allows user-defined blocks to be added.

Node Components

Bundle Protocol Agent.

The *Bundle Protocol Agent* (BPA) offers BP and DTN specific services. It executes procedures of the BP. For example, communication between Application Agent and Convergence Layer Adapter (see below) is managed. The BPA also constructs bundles for the Application Agent.

Application Agent.

The interface between the BPA and an application is defined as an *Application Agent* (AA). A generic AA needs the ability to receive incoming bundles and compose outbound bundles for user applications and services. Furthermore, an EID must be assigned for local bundle delivery.

Convergence Layers.

Bundles are exchanged over connections between nodes of different types and characteristics, and connections are unidirectional or bidirectional, or vary in transmission speed and bandwidth. Depending on the connection technology used, more or less complex protocols are required for delivery, called *Convergence Layer (CL) Protocols* (CLP). A *Convergence Layer Adapter* (CLA) is an implementation of a CLP. There are two CLPs defined by the IETF DTN group to exchange bundles over a TCP connection, the bidirectional TCP Convergence Layer Protocol (TCPCL) [Sip+19] and the unidirectional Minimal TCP Convergence Layer Protocol (MTCP) [Bur19]. In addition to transport layer CLs, there are approaches based on other technologies, e.g., DTN2 defining a Bluetooth and a serial CL, or IRB-DTN featuring an e-mail CL.

5.4.4 DTN7

In this section, we present the design and implementation of DTN7.

Requirements Analysis

There are several requirements that should be satisfied by DTN software. First, DTN software operating on a variety of laptops, smartphones, and routers should run on several hardware architectures (e.g., x86, ARM, and MIPS), based on the most popular operating systems (e.g., Linux, macOS, and Windows). Second, the individual components of the DTN software should be exchangeable. For example, there is the need to support different storage backends, CLAs, and DTN routing protocols. A suitable programming interface enabling concurrent execution is required for the interaction of components. Furthermore, a CLA implementation is required as well as a peer discovery mechanism to enable automatic establishment of connections between nodes. Finally, applications should to be independent of the DTN software, to allow easy creation of further applications and tools. Thus, a convenient interface between the DTN software and applications is required.

Implementation Decisions

As a result of these requirements, we selected the Go programming language[28] to develop DTN7. Go offers a large standard library and is rather developer-friendly. Its strengths are the simple creation and integration of programming libraries. Moreover, Go enforces good style guides and clean code plus provides memory-safety guarantees to increase security and stability of written programs. Thus, Go makes maintaining code and bringing in new developers very easy. The source code including all required dependencies are compiled into a single, static executable, removing the need for interpreters or further libraries. Furthermore, the Go compiler allows simple (cross-)compilation for many operating systems and processor architectures. The concept of concurrency is implemented in Go through the interaction of Goroutines and Channels; concurrency was one of the design priorities of the language designers.

To support exchangeability of DTN7's components, we structured our implementation into *Bundles* and its corresponding *Store*, *Convergence Layer Adapters*, *Peer Discovery*, the *Application Agent*, *Routing*, and the *Core* package needed to connect the individual packages. The modules in the these packages are designed as generic interfaces and example implementations, e.g., there exists an interface for routing in general and an epidemic routing implementation. We decided to use MTCP for exchanging messages between two DTN7 nodes due to its simplicity. A third party application can also use parts of DTN7 as a library to, e.g., create and serialize bundles via the corresponding package. To make programming of applications against these interfaces simple and programming language independent, we decided to use a RESTful API.

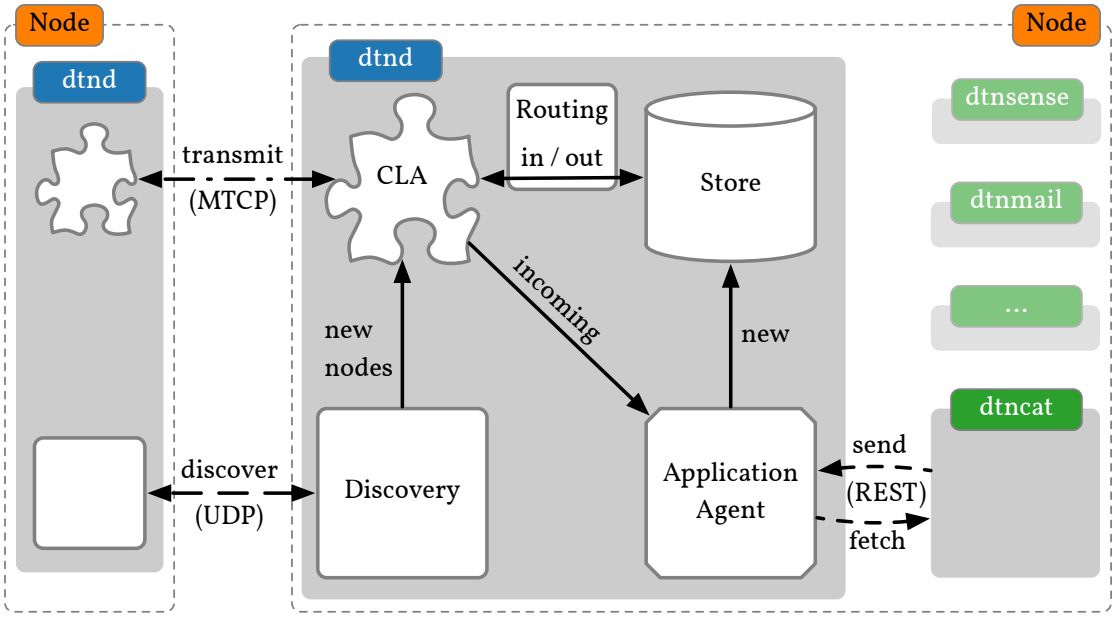


Figure 5.24: Architecture and data flow in DTN7

DTN7 Architecture

Fig. 5.24 shows the modules of DTN7 and their interaction. The arrows indicate the way a bundle is internally processed in DTN7. The links between two distinct DTN7 nodes are shown by both an active CLA and the Discovery on the figure's left hand side. Multiple client connections to the AA from within the node are delineated on the node's right hand side.

To store bundles locally, a serialized version as defined in BP7 is written to the file system. A central index of all known bundles manages their meta-data and links point to information of the specific file. This index supports a fast lookup of bundles. The module providing this functionality is called *Store*.

In DTN7, an AA is implemented as a RESTful Web API to support both dispatching and fetching of bundles. The API does not interact with entire bundles, but only with a subset of its fields. This allows a client to send a new bundle by only supplying the destination EID and a payload. Such a request can easily be created from the command line or possible third-party software. When fetched over the API, selected fields of those bundles are returned and the bundles will be removed from the store afterwards.

The concept of different CLs and their CLAs is also present in DTN7's architecture with an implementation of MTCP. Based on a specific CL's characteristics, bundles might be transferred in a uni- or bidirectional way. Thus, a CLA in DTN7 must supply one or multiple modules for inbound and outbound bundle processing. The unidirectional MTCP is designed using modules for sending and receiving bundles.

[28] `https://golang.org`

To support connections in dynamic networks, a *Peer Discovery* mechanism is provided. It announces a node's existence and listens for potential neighbors. This discovery mechanism broadcasts all of the node's CLAs continuously and notifies about received CLAs.

The previously defined components are linked together within DTN7's *Core* package. A central processing pipeline consumes both newly created and inbound bundles. Within this pipeline, a bundle will be marked to be delivered to a subset of known CLAs, to a local AA or to be discarded for later processing or even removed. The Core's internal links, visualized in Fig. 5.24, are related to the concept of a BPA, and serve as an interface between CLAs and the AA.

Every bundle that is not addressed to a particular node will be forwarded over one or multiple CLAs to neighboring nodes. The decision about which CLAs to select is made by a routing algorithm. To support the use of different routing algorithms, a generic interface needs to be informed about inbound bundles and, furthermore, a tight cohesiveness to the core is required. DTN7 implements an epidemic routing module, which is notified about received bundles, to memorize both sender and receiver. Before dispatching, the epidemic routing algorithm compiles a subset of known connections which have not received this bundle yet.

Finally, DTN7 is also intended to be used as a library and allows fast development of DTN applications. In particular, bundle package creation, serialization, and deserialization might be useful in other software.

Resulting Programs

DTN7 contains a DTN daemon, referred to as `dtnd` in Fig. 5.24, for storing and exchanging bundles and interfacing with applications. Currently, an example DTN application (`dtncat` in Fig. 5.24) for sending and receiving bundles, implemented as a command line tool, is included. `dtnd` initializes the previously defined modules according to the configuration provided by the user. `dtncat` processes user input, which is handed over to `dtnd`'s AA RESTful interface. The input is then encapsulated inside the Payload Block of a newly created bundle by `dtnd`. This bundle's Primary Block will be populated with basic defaults, like disabled CRC, and a delivery report request. As shown in Listing 5.1, `dtncat` is called by passing parameters on the command line. The first option selects between receiving or sending bundles. The local `dtnd`, running the RESTful API, is addressed by the second parameter. When sending new bundles, the content is read from the standard input.

```
# Sending a bundle
$ dtncat send http://localhost:8080 dtn:s2 <<< "3782␣lx"

# Retrieving a received bundle
$ dtncat fetch http://localhost:8080
```

Listing 5.1: `dtncat` example

5.4.5 Experimental Evaluation

In this section, we experimentally evaluate DTN7 and compare it with other DTN software.

Emulation Environment

To evaluate DTN7 in a realistic manner, we emulated up to 64 nodes in the network emulation framework *Common Open Research Emulator* (CORE) [Ahr+08]. CORE can emulate nodes using Linux namespaces to allow the execution of native binary programs, which is not possible with purely simulation-based approaches like NS-3 [RH10; Sch+11b]. All experiments were performed on Intel Xeon E5-2698 CPUs with 80 cores at 2.20 GHz and 256 GB RAM. To execute the total number of 1,440 experiment runs, we used MACI, a framework for extensive and reproducible experiments [Frö+18].

DTN Software.

We compared DTN7 with three popular DTN software solutions. *Serval*[29] is a software suite centered around protocols designed for infrastructure independent communication [Gar11]. To be able to transfer files in intermittently connected networks, Serval relies on Rhizome, a custom DTN bundle protocol with epidemic routing. In our evaluation, we used the latest stable Serval release, which is from April 2016, since the recent development version has stability issues. *IBR-DTN*[30] is an implementation of BP Version 6, aimed to be lightweight and fast [Doe+08]. For comparability, we use the epidemic routing extension instead of the default PRoPHET protocol used by IBR-DTN. We use the current HEAD of the git repository to include the latest bug fixes. *Forban*[31] is mainly used as a local peer-to-peer file sharing application using an epidemic routing protocol based on HTTP. We used the latest HEAD of the git repository, but had to introduce our own patches to make Forban usable.

Payload Sizes.

DTN software is used in multiple applications and scenarios. Serval, e.g., offers the SMS-like application MeshMS for short text messages. IBR-DTN can be used in environmental monitoring, where transmission of short audio recordings or images might be required. Therefore, we selected four different file sizes, representing a wide range of possible applications. All files were generated randomly with the same seed for reproducibility in six sizes:

- *64 KiB* for compressed images or map data;
- *1 MiB* representing small images or short audio recordings;
- *5 MiB*, e.g., smartphone images and audio recordings;
- *25 MiB* representing longer audio recordings or short videos;
- *50 MiB* for HD videos typically recorded by smartphones;
- *100 MiB*, e.g., 4k smartphone videos [Tro+15; Sch+11a; Bau+16].

[29] `https://github.com/servalproject/serval-dna/tree/batphone-release-0.93`
[30] `https://github.com/ibrdtn/ibrdtn`
[31] `https://github.com/adulau/Forban`

Network Topologies.

We used a chain topology of three different lengths, where nodes are connected pairwise, to benchmark the different DTN software systems. The first node is sending a bundle destinated to the last node in the chain. To get the baseline performance of the interacting components, a chain of two nodes was used. We measured the time it takes to read the data, serialize the bundle, send it over the network, deserialize it at the receiver and deliver it to the application. With 32 nodes, the forwarding capabilities were investigated. For an even larger scenario, we used 64 nodes, to evaluate how the DTN software systems behave when node numbers increase. We used a bandwidth of 54 MBit/s to match the speed of an IEEE 802.11g network.

Measurements.

To measure CPU utilization for each process on every node, we used *pidstat*, which is part of the *sysstat* package[32]. Additionally, *bwm-ng*[33] was used for network statistics per node and network interface. Finally, every used DTN software logged both the timestamp of sending and receiving bundles, such that a detailed analysis of transmission time and network distribution can be performed.

Results

Transmission Times.

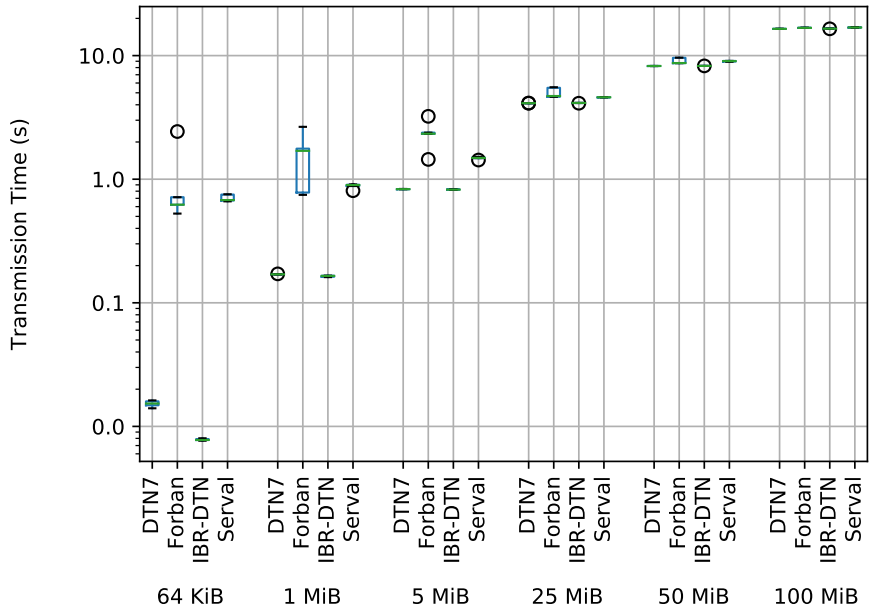


Figure 5.25: Bundle transmission time for the 1-hop topology and different payload sizes

[32] http://sebastien.godard.pagesperso-orange.fr/man_pidstat.html
[33] https://github.com/vgropp/bwm-ng

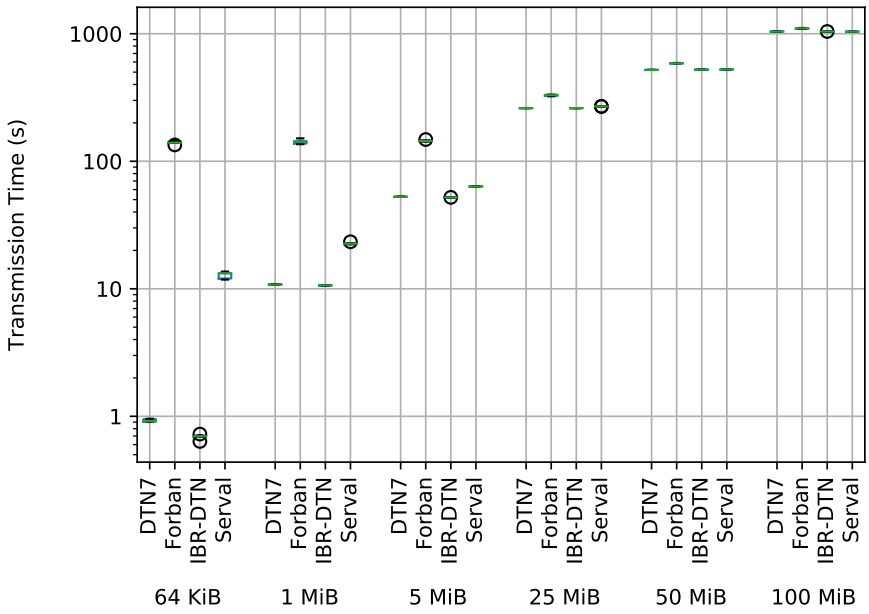


Figure 5.26: Bundle transmission time for the 64-hops topology and different payload sizes.

Figs. 5.25 and 5.26 show the bundle transmission times on the y-axes and payload sizes on the x-axes for the 1-hop and 64-hops topologies, respectively. Regardless of chain length and file size, DTN7 and IBR-DTN are always the fastest DTN software systems. The larger the files become, the transfer times of all DTN systems converge. This is due to the network configuration. All DTN systems manage to completely fill the 54 Mbit/s available, which is easier to achieve with larger files. As a result, the transfer times for large files hardly vary at all.

For a single hop, Forban and Serval take about the same time for transmitting files (e.g., about 0.6 seconds for 64 KiB files), but Forban shows a higher variance. For longer chains and files below 50 MiB, the differences between Forban and Serval are more noticable. DTN7, however, is still up to 140 times (64 KiB over 1 hop) faster than Serval. Particularly in chat or text based applications, the speed advantage of DTN7 can be crucial if a message arrives below 0.01 seconds rather pdfcthan after one second.

These results indicate that both BP6 and BP7 have a relatively small protocol overhead compared to the protocols used by Serval and Forban, which is especially noticeable for small files. The larger the files or the longer the chain, the less weight the low protocol overhead carries. Furthermore, it is also remarkable that DTN7, which is written in Go, does not take longer to transmit larger files from end to end in the chain, although IBR-DTN is implemented in C++ and optimized for speed. In terms of transmission speeds, Forban takes longer than the other DTN software systems, although differences get smaller the bigger the files are. One explanation is that Forban has a pull-based approach where it actively downloads new bundles after an announcement was received. Therefore, the announcement interval is a natural barrier. If quick data exchange is necessary, the other solutions provide better performance.

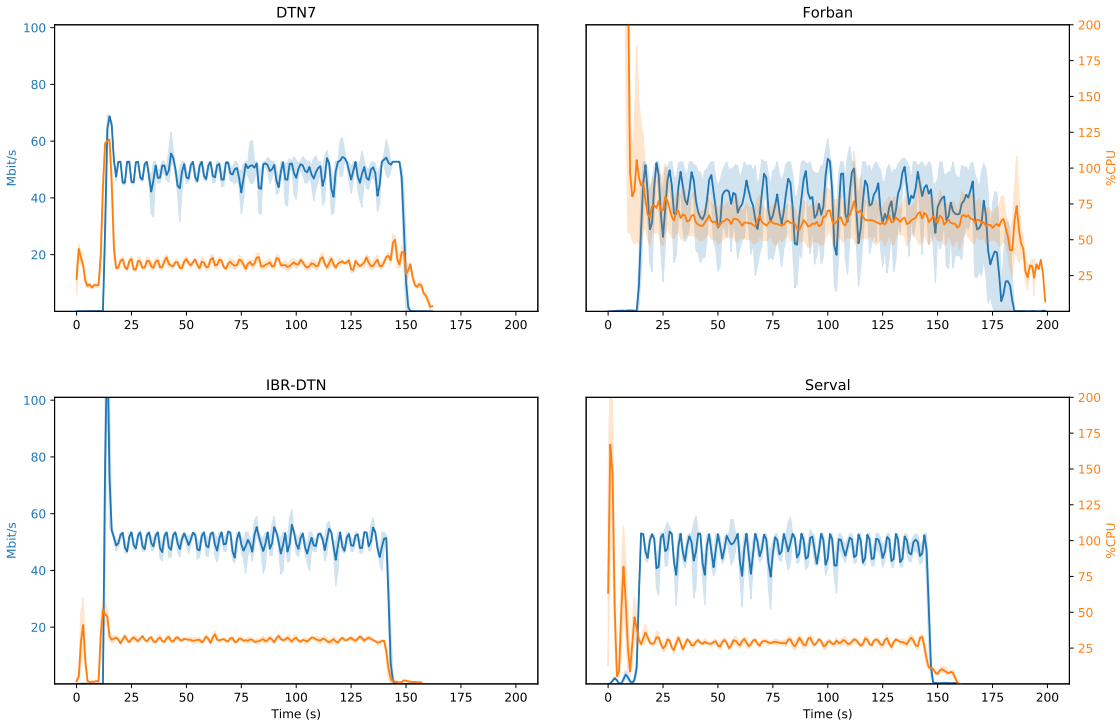


Figure 5.27: CPU and network usage for transmitting 25 MiB over 32 hops.

CPU Usage and Network Utilization.

Fig. 5.27 shows CPU usage and network utilization for transmitting 25 MiB over 32 hops. On the x-axes, the time for the entire experiment in seconds is shown, the left y-axes denote the network usage in Mbit/s and the right y-axes show the CPU usage in %, both of the entire network. The bold graphs denote the sum over all nodes, averaged over all experiment repetitions. The shaded areas denote the error band.

DTN7 requires about 34.3% of the available CPU (standard deviation of 16.7%). At the beginning of an experiment, DTN7 shows a short peak in CPU usage resulting from the first node, where the file is converted to base64, sent to the DTN7 AA, which decodes the file again, packs it into a bundle, and starts the transmission. Further nodes only have to retransmit the bundle and do not require the steps mentioned above. Forban uses about 163.1% CPU (646.3%). Forban shows a small peak at the start of the experiment, indicating the overhead when starting its daemons, where four Python interpreters have to be started. Additionally, the file has to be hashed at the beginning of the experiment. Serval consumes 29.3% (24.6%) CPU. Serval has an additional hashing step, which results in higher CPU load at the start of the experiment. With only 26.9% (13.1%), IBR-DTN is the most efficient tested DTN software in terms of CPU usage.

In terms of network usage, DTN7 reaches about 42.0 Mbit/s (19.7 Mbit/s) for transmitting bundles from node to node, while Forban achieves about 32.8 Mbit/s (22.8 Mbit/s). IBR-DTN and Serval achieve 42.3 Mbit/s (23.7 Mbit/s) and 39.5 Mbit/s (20.0 Mbit/s), respectively. Although the theoretical total network load for the entire network can be up to 1.674 Gbit/s, the tested DTN software systems used only the maximum bandwidth per link, which is 54 Mbit/s, in peak

situations. This indicates that every DTN software needs to receive the entire bundle before transmitting it to the next node.

To summarize, DTN7 requires slightly more CPU utilization than IBR-DTN and Serval, but has the advantage of transmitting files faster than all other DTN systems in most cases, as shown in Section 5.4.5.

5.4.6 Summary

We presented an open source DTN implementation, called DTN7, of the recently released Bundle Protocol BP7, written in the Go programming language. DTN7 is designed to offer extensibility and supports multiple use cases, such as enabling communication in emergency and disaster scenarios or providing connectivity for rural areas. Furthermore, we presented results of a comparative experimental evaluation of DTN7 and other DTN systems including Serval, IBR-DTN, and Forban. Our results indicated that DTN7 is a flexible and efficient open-source multi-platform implementation of the most recent version of BP7. DTN7 is not directly suitable for categorization within the framework of the thesis, but thanks to the modular development method it forms the basis for the further development of smart systems.

5.5 ProgDTN: Programmable Disruption-tolerant Networking

5.5.1 Introduction

Originating from developments related to the exploration of outer space, *disruption-tolerant networking (DTN)* has found its way into numerous terrestrial applications, e.g., in scenarios where communication networks are destroyed or disrupted and cannot be repaired for days. Apart from natural disasters, it may be hard for people to communicate via mobile devices in remote areas without a deployed telecommunication infrastructure. When end-to-end connectivity is not available, DTN can be utilized to keep communications going without the need for traditional infrastructures. However, utilizing DTN for communication requires custom network protocols, since protocols of the widely used TCP/IP stack are not well suited for such challenging situations. Therefore, several routing algorithms were developed for DTNs. Some are targeted at general purpose applications, i.e., they rely on the connectivity inside the network or are based on conventional routing schemes, such as link-state routing. Other DTN routing algorithms are constructed for specific scenarios, such as the movement of people during a workday or emergency situations. We argue that each DTN routing algorithm is designed for a specific scenario in which it achieves very good performance, but has limited performance in other scenarios.

Conventional network protocols of the TCP/IP stack suffer from the same limitations, which motivated the introduction of *programmable networks* in the literature. For example, Software-defined Networking (SDN) offers programmable means of configuring networks using customized algorithms for processing and routing. Typically, this leads to optimized performance in terms of latency, throughput, and/or resilience. We argue that DTN should also take advantage of network programmability to profit from its benefits like time and cost savings, reduction of

human error, customization, and improved performance. However, the SDN approach cannot directly be transferred to DTN, because SDN, in general, requires a coordinating entity that deploys (programmable) rules to network nodes. In most cases, a coordinating entity is not present in DTNs.

In this section, we present *ProgDTN*, a novel approach to support *programmable disruption-tolerant networking* by allowing network operators to program a node's routing behavior based on DTN bundle metadata, additional bundle context, and node context. Node and bundle context can be used to reflect the specifics of a particular scenario, e.g., the speed of a node in a mobile scenario, the battery level of nodes in a scenario without fixed power supply, or geographic information of the context of a bundle. ProgDTN consists of a programming interface that allows a network operator to program the routing algorithm without knowledge of the router's interior workings. ProgDTN's implementation is based on DTN7 [Pen+19], a flexible and efficient open-source, platform-independent implementation of the Bundle Protocol version 7 [BFB22], and uses JavaScript as the programming language, because it is widely used and easy to understand. In our experimental evaluation, we compare ProgDTN to four existing routing algorithms. We demonstrate that a programmable DTN routing algorithm tailored to a specific scenario achieves excellent results in terms of up to 99.9% delivery ratio while reducing unnecessary transmissions by 92.9% compared to other state-of-the-art DTN routing algorithms in an emergency response scenario. We achieve a low delivery time of bundles (1 – 15 seconds) and a low overhead in terms of CPU utilization and routing decisions. Our contributions are:

- We present ProgDTN, a novel approach for programmable DTN routing.
- We show that using ProgDTN, network operators can tailor routing algorithms to their individual scenarios by incorporating context information.
- We present a comparative experimental evaluation of ProgDTN, achieving excellent results in terms of delivery ratio, delivery times, and overhead.
- We make ProgDTN's implementation, our scenario-specific routing algorithm,[34] and code/data fragments of our experimental evaluation[35][36] available under permissive open-source licenses.

Parts of this section have been published in Markus Sommer, Jonas Höchst, Artur Sterz, Alvar Penning, and Bernd Freisleben. "ProgDTN: Programmable Disruption-tolerant Networking." in: *International Conference on Networked Systems (NETYS)*. Springer. May 2022. DOI: 10.1007/978-3-031-17436-0_13.

5.5.2 Related Work

This section gives a brief overview of related work on context-aware routing in general and context-aware routing in DTN.

[34] https://github.com/umr-ds/dtn7-go/tree/progdtn
[35] https://github.com/umr-ds/progdtn-evaluation
[36] https://dshare.mathematik.uni-marburg.de/index.php/s/8k6XZgKJp9kTMPS

Context-aware Routing.

Apart from using information about the network topology, context-aware routing algorithms use additional information to make routing decisions. Here, we focus on context-aware routing protocols for wireless mobile ad hoc networks (MANETs).

The History-based Routing Protocol for Opportunistic Networks (HiBOp) uses social information, such as club memberships or home addresses to infer the likelihood of encounters and improve routing decisions [Bol+07]. Dynamic Social Grouping-based Routing (DSR) is a routing algorithm that harnesses social grouping for efficient routing in ad hoc networks [Cab+10]. The Inheritance Inspired Context-Aware Routing Protocol (IICAR) follows a biology-inspired approach to routing based on Mendel's laws of inheritance [Ban+19].

The protocol proposed by Biswas et al. [Bis+18] uses the properties of ad hoc wireless networks. It maintains static context, such as node and interface types and social information, and dynamic context, e.g., geolocation, channel quality, and encounter frequency. The information is used to calculate utility scores, combined using a routing metric to determine a delivery probability for each message. Messages are forwarded to a) the closest short-range neighbor and b) a long-range neighbor selected based on the delivery probability and the distance.

The algorithm proposed by Errouidi et al. [Er-+17] uses context information to improve resilience in MANETs. It employs a fuzzy logic system for three context metrics, a node's remaining energy storage, distance between peers, and node mobility, to judge its "stability". This allows it to improve routing decisions by avoiding unstable nodes, which improves network stability and delivery metrics.

Context-aware Delay-tolerant Routing.

The Context-Aware Adaptive Routing (CAR) protocol harnesses context information for routing decisions DTN and combines synchronous and asynchronous transmission [MHM05; MM09]. Messages are transmitted synchronously using a distance vector routing approach if the recipient is located in the same neighborhood. If no direct connection between sender and recipient exists, asynchronous (DTN) transfer is used, where a node computes delivery probabilities of the directly connected nodes based on its context information. It is up to the network provider to define concrete attributes, utility functions, and weights for the generic approach.

The Sensor Context-Aware Routing (SCAR) protocol for distributed sensor networks [MM06] is based on the CAR protocol, omitting the distinction between synchronous and asynchronous transmissions. The approach only handles a single specific use case and does not attempt to provide a general routing algorithm.

A further routing scheme that competes with CAR was proposed by Johari et al. as Context-Aware Community Based Routing (CACBR) [JGA13]. Message forwarding is based on a combination of network and context parameters, such as communities a peer belongs to, message delivery and forwarding history, and available buffer space and battery level.

In general, schemes that attempt to harness social and community relations are relatively common; another example is the Socially-Aware Adaptive Delay Tolerant Network (DTN)

routing protocol by Ullah and Qayyum [UQ22]. It utilizes a metric called degree centrality to estimate how well a node is embedded in a community to drive forwarding decisions.

Beak et al. [BSC18] propose a version of the PRoPHET routing algorithm, improved by using context information. While the authors show that their proposed protocol outperforms regular PRoPHET, it still relies significantly on the underlying protocol, and the use of context information is limited.

Another approach was proposed by Rosas et al. [RGH20]. It is not focused on designing a routing algorithm that includes context information, but instead measures the performance of different algorithms under different context values and then uses future context information to choose the optimal one.

To the best of our knowledge, no attempt has been made in related work to create a general-purpose, context-aware, programmable DTN routing system.

5.5.3 ProgDTN Design

In this section, we present the design of ProgDTN, including DTN fundamentals, system requirements, context information, and ProgDTN's architecture.

DTN Fundamentals

The data transmission unit of the *bundle protocol* is a *bundle*, which consists of multiple *blocks*. Each bundle must contain a *primary block* containing basic metadata, such as the bundle's ID, sender and recipient IDs, and a *payload block* that carries the bundle's payload. A bundle can also include an arbitrary number of *extension blocks*. While the DTN standard specifies several extension block types, it allows implementations to specify additional types.

A node in DTN operates in a store-carry-forward manner, i.e., when a bundle is received, it is stored in local, long-term storage, from where it will be regularly forwarded. When forwarding, the network daemon invokes the configured routing algorithm to select a subset of currently connected peers to which the bundle should be forwarded. The actual peer-to-peer connection is abstracted in a so-called *convergence layer* (CL) that may use any lower-layer communication protocol to achieve data transmission. ProgDTN is designed to be entirely independent of any specific communication infrastructure and works with any standard-compliant convergence layer.

System Requirements

ProgDTN's goal is to allow network operators to develop scenario-specific routing algorithms without modifying the DTN software itself. Furthermore, changes to the routing algorithm should not require recompilation of the DTN software to reduce the complexity of deploying new or adjusted routing algorithms. Instead, the forwarding rules are loaded at startup from a provided script file and interpreted by the DTN software. In this way, algorithms may be swapped by restarting the DTN software and giving it a different file to load. To specify these

forwarding rules, we use a general-purpose programming language and embed an interpreter into the DTN software. This allows maximum flexibility without having to learn a new domain-specific language.

Context Information

ProgDTN belongs to the class of context-aware routing algorithms, i.e., routing decisions may be based on additional information of the environment in which the network exists. *Context* refers to any information about the nodes, bundles they are transmitting, or any other information that the network operator may deem helpful. ProgDTN does not put any semantic restrictions on context information, except that each piece of information needs to be uniquely named. Instead, we provide network operators with a generic, powerful interface for generating and processing relevant context information.

Context Types. ProgDTN distinguishes between two classes of context information, *bundle context* and *node context*. Bundle context is any additional information attached to an otherwise normal bundle, e.g., the physical location of the bundle's recipient, information about the originator or the recipient, and really anything that the network operator might think of. On the other hand, node context is information about a specific node, such as the node's location, the node's battery status, and its connectivity status. This information is not usually attached to other bundles, but if it must be communicated to other nodes, the node broadcasts it using a special context bundle.

Context Generation. The naive approach for generating context information would be to have the DTN software itself generate the necessary information. This would, however, violate the ease of use goal, since it would require modification of the DTN software's code for each scenario. ProgDTN adopts an approach where context generation is left up to external programs. For this purpose, the DTN software exposes an interface through which context information can be injected. This interface should be based on a widely used communication protocol/architecture to ensure ease of use.

Context Transmission. Since context information needs to be attached to a bundle, and since it may be helpful for nodes to be able to exchange their contexts, we defined a custom extension block to carry context information. This extension block may either be attached to an existing bundle or be used to exchange context data with peers by sending a special *context bundle*. Extending a regular bundle with a context block allows nodes to use this information when making forwarding decisions.

ProgDTN Architecture

Fig. 5.28 shows the architecture of a DTN deployment utilizing ProgDTN. The yellow circles depict mobile DTN nodes, whereas the violet circles show data sources in a hypothetical scenario. Each node is identified by the prefix `dtn:` followed by a name, such as `n1`, used as

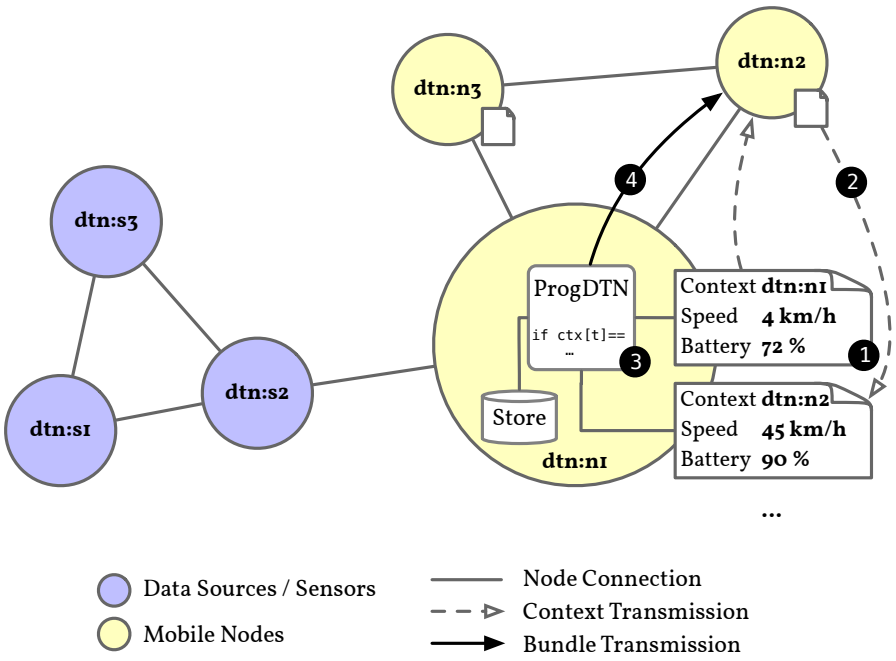


Figure 5.28: Architectural overview of a DTN deployment utilizing ProgDTN

the address for bundle transmissions. The lines between the circles show connections between the particular nodes. Node `dtn:n1` is used to visualize further concepts.

In general, when a bundle enters a DTN node, either by being created or received from a peer, it is placed into the node's local on-disk storage (`store` in Fig. 5.28). The bundle is then passed to the configured routing algorithm to select peers for forwarding. ProgDTN takes the position of the routing algorithm, which is visualized by the box entitled `ProgDTN`. It takes the particular bundle, which can itself have attached context information, and the context information for its node (shown as ① in Fig. 5.28), as well as the peer's context (②). The context in this example is the speed and battery level of the transmitting node and for node `dtn:n2`. The interpreter (③) then executes the routing script provided by the network operator, which must be aware of the available context data, and filters the list of connected peers to select the subset for forwarding. Once the routing program has returned the list of selected peers, the DTN software transmits the bundle to the selected peers (④).

5.5.4 ProgDTN Implementation

ProgDTN's implementation is based on `dtn7-go`, a powerful DTN software suite developed in the DTN7 project[37]. `dtn7-go` implements the most recent draft of the bundle protocol (BP) version 7 [BFB22] in the GO programming language [Pen+19].

Fig. 5.29 shows the components of `dtn7-go`. The two dotted boxes indicate two DTN nodes, and the arrows between them show their interaction, data transmission (`transmit`), and peer

[37] `https://dtn7.github.io/`

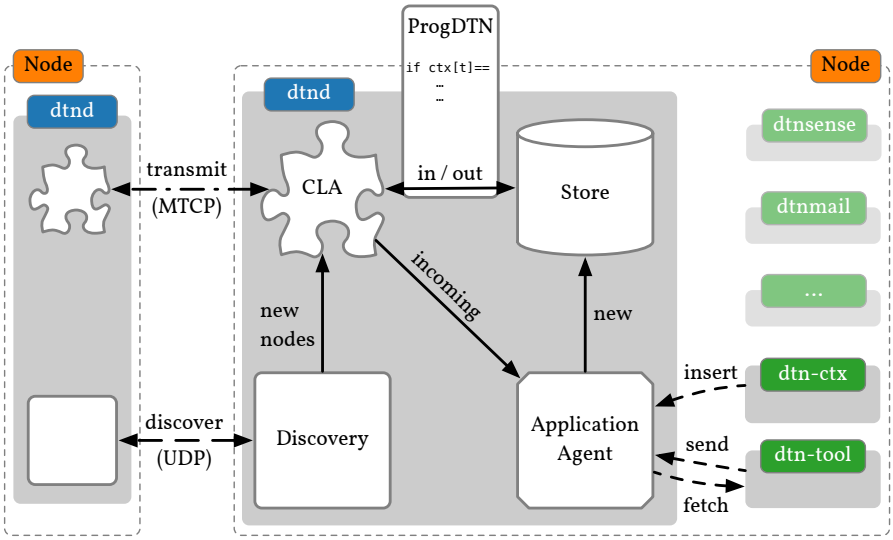


Figure 5.29: `dtn7-go` with the ProgDTN implementation between CLA and Store

discovery (`discover`). The arrows within the `dtnd` box visualize the data flow of bundles within a node. The right-hand side shows multiple tools that use an application agent (AA) to insert data to or retrieve data from `dtn7-go`. The AA then stores the received data in the node's local store. The bundle protocol abstracts peer-to-peer communication using convergence layer adapters (CLAs), which can use a variety of protocols and technologies such as TCP, UDP, or even e-mail sent over Bluetooth. Each CLA exposes a defined API to the daemon, which can then transparently send & receive messages without having to bother with data serialization or transmission. To support connections in dynamic networks, `dtn7-go` uses a peer discovery mechanism that continuously broadcasts information on all of the node's CLAs and notifies the daemon about newly discovered peers. For forwarding decisions, `dtn7-go` includes an API that can be used to implement routing algorithms. Any *Go* datatype that satisfies this interface can be used as the router, and `dtn7-go` ships with various established routing algorithms. ProgDTN consists of an implementation of that interface and a set of extensions for receiving and storing bundle and node context.

Using JavaScript for Programmable Routing

We decided to use JavaScript as our general-purpose programming language for programmable DTN routing due to the following reasons: (a) JavaScript is very popular; according to a 2021 survey conducted by Stack Overflow[38], nearly 65% of developers work in JavaScript, giving it a roughly 17% lead over the second-placed general-purpose language Python at approximately 48%; and (b) JavaScript can be embedded into the GO programming language; there are several JavaScript interpreters written in GO, e.g., `goja`[39], which implements the ECMAScript standard version 5.1 with some additional features.

[38] `https://insights.stackoverflow.com/survey/2021`
[39] `https://github.com/dop251/goja`

Thus, a routing algorithm is specified in a JavaScript file that gets loaded during startup of `dtn7-go` and compiled to bytecode representation for faster execution. Whenever a routing decision is made, the compiled JavaScript routing algorithm is invoked, which invokes a virtual machine (VM) able to interpret the JavaScript bytecode.

Programmable Routing Decisions

Any custom routing algorithm has to comply with the following API. First, the JavaScript code receives the bundle itself passed as a JavaScript object [40], as well as the bundle ID, represented as a string. Second, the ID of the bundle's source (as a string) and a list of strings representing the IDs of all currently available peers are passed to the routing algorithm. The final pieces of information are the bundle context, the node's context, and the context of all available peers. Whenever the algorithm receives context data, this data is encoded as JSON, a data serialization format that works well with JavaScript. The bundle's context can be updated using a callback function whenever this may be necessary. If logging is required, it is possible to use the passed `loggingFunc` function to write an arbitrary string to the `dtn7-go`'s logs. The algorithm may then use any or all of this data to perform the actual forwarding decision. Any JavaScript code can be executed here, including third-party libraries. ProgDTN does not place any restrictions on the possible context data; it is up to the network operator to be aware of runtime requirements. Finally, the routing algorithm must return a list of node IDs to which the bundle should be forwarded. `dtn7-go` then takes care of forwarding the bundles to the chosen nodes using the corresponding CLAs.

Providing Context

To provide context information about the local node, we implemented a REST interface that receives information formatted in a key-value manner, where the value contains arbitrary data formatted as JSON. The context information is then saved in a global dictionary so that newer data for a given peer overwrites existing data. While the local node's context information is vital for routing decisions, so is its peers' context. Therefore, whenever two peers connect, they exchange their context information. Finally, for bundle context, the provided REST interface for bundle submission is extended to add context information to the bundle during its creation, again as arbitrary key-value pairs.

[40] The description of the bundle data structure is omitted for brevity. We refer to `https://github.com/dtn7/dtn7-go/blob/d3b5e62a7f89994ececf98978bae499f32cc920f/pkg/bpv7/bundle.go` for further information.

Parameter	Values
Bundles per Node	10, 50, 100
Payload Size	1 kB, 1 MB
Routing Algorithm	Epidemic Routing, Binary Spray & Wait, DTLSR, PRoPHET, ProgDTN Epidemic, ProgDTN Binary Spray & Wait, ProgDTN

Table 5.8: Evaluation Parameters

5.5.5 Experimental Evaluation

Emulation Environment

Network Emulation.

To perform a large number of experiments, we used the *Common Open Research Emulator (CORE)* [Ahr+08], an open-source network emulator[41]. CORE supports the execution of native binaries without re-implementing protocols, i.e, real-world code can be executed. Furthermore, we used the MACI experimental orchestration framework [Frö+18] to schedule a large number of experiments. All experiments were executed on an AMD EPYC 7742 server with 128 physical cores and 1 TB RAM, which executed up to 3 experiments in parallel.

Network Topology.

We use a network topology that simulates a disaster scenario involving three parties, *civilians*, *responders*, and a *coordinator*. The scenario consists of 31 nodes arranged in a 2-circle topology, with the singular coordinator located in the center, five responders arranged in a circle around the coordinator, and 25 civilians arranged in the outer circle. Each responder is connected to 5 civilians, and the civilian clusters are connected on their edges. Each civilian sends bundles addressed to the coordinator, simulating information moving up the chain of command. The coordinator produces broadcast bundles that are supposed to be received by all civilians, which simulates announcements by the authorities to the population. The experiments simulate a network with a bandwidth of 54 MBit/s, 20 ms of delay, and a range of about 40 meters.

Experimental Parameters.

All experiments are uniquely defined by a set of four parameters, summarized in Table 5.8 and discussed below. In total, 210 experiments were executed for one hour each. *Bundles per node* determine how many bundles each civilian and the coordinator send to the network. To avoid every node sending its data simultaneously, bundles are sent at (uniformly distributed) random times throughout the experiment. We used two different *payload sizes*, 1 kB and 1 MB, to mimic different use cases, with 1 kB serving as a stand-in for text messages and 1 MB being

[41] `https://coreemu.github.io`

around the size of a small image. One of the two payload sizes was used for all nodes during every experiment. To maintain reproducibility and to reduce the chance of unfavorable initial conditions, each experiment was re-run five times with pre-determined PRNG-seeds.

Seven routing algorithms are used for comparison: *Epidemic Routing* [VB00] is the most widely used routing algorithm in DTN, sending bundles to all peers.

Binary Spray & Wait [SPR05] is a modified version of Spray and Wait. A node holds n copies of a bundle, half of which are transferred to the first peer. The second peer then receives half of the remaining copies and so on. Each node that receives multiple copies of a bundle proceeds in the same way. A node will only forward the bundle to its intended recipient when only a single copy remains. Spyropoulos et al. [SPR05] suggest a value of 10 as a reasonable initial multiplicity, which we adopted in our experiments.

PRoPHET [LDS04] exploits the fact that in DTNs, nodes usually encounter each other more than once. Whenever two peers connect, they compute a probability of meeting again in the future. This probability declines over time if these particular nodes do not meet again. A node will forward a bundle to another node if the receiving node's probability of meeting the bundle's recipient is higher than the forwarding node's probability. We used the same parameters as the authors in their original paper for calculating delivery probabilities.

Delay-Tolerant Link-State Routing (DTLSR) [DF07] other than classical link-state routing once a link is lost, it is not immediately removed from routing considerations, but rather "tagged" with the time since the disconnection. When the routing table is computed, this time is interpreted as a link cost of Dijkstra's algorithm to find the shortest path between the current node and the destination.

ProgDTN Epidemic / Binary Spray & Wait are re-implementations of the respective algorithms in ProgDTN, which serve primarily to compare computational overheads. All parameters are the same as in the native implementations.

ProgDTN Emergency is a custom algorithm implemented using ProgDTN and tailored to the given scenario of our evaluation, where data only flows in two "directions": from civilians to the coordinator, or vice versa. Whether a node will forward a bundle to another node depends on three factors: (a) node type (coordinator, responder, civilian), provided at startup, (b) peer type, received by a node via a context bundle and (c) bundle type (unicast to coordinator, or broadcast to all civilians), carried by a bundle in a context block attached at bundle generation time. Unicasts (i.e., bundles from the civilians to the coordinator) are only forwarded along the inward direction, from civilians to responders to the coordinator, while civilians do not send their bundles to each other. The same applies to responders. Broadcasts flow outward, i.e., from the coordinator to the responders to the civilians; civilians distribute messages among each other. In both cases, responders do not forward bundles among other responders, but only serve as relays between civilians and the coordinator.

Results

We consider six metrics divided into two categories: network utilization and an overhead analysis. The network utilization metrics are the percentage of bundles successfully delivered,

the duration of delivery, and the load generated in the network. Our overhead analysis considers the time to decide to whom a bundle should be forwarded, the percentage of bundles that do not carry a payload (metadata or context bundles), and how heavily a node's CPU is utilized.

Delivery Ratios.

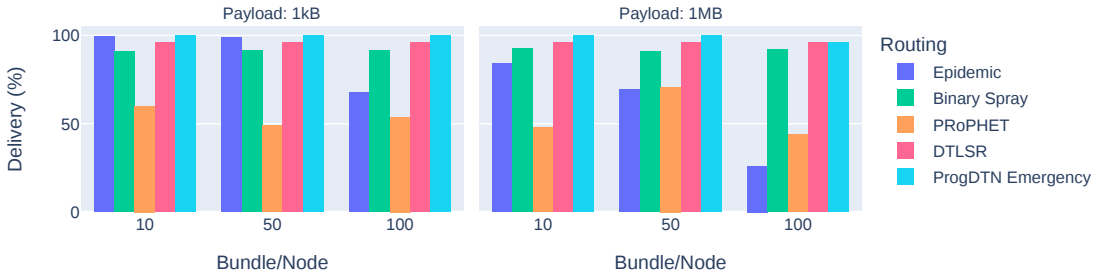


Figure 5.30: Ratio of successfully delivered bundles for different parameters

Fig. 5.30 shows the *delivery ratio*, i.e., the percentage of sent bundles that reach their destination. Each group on the x-axis represents a set of bundles per node, the y-axis shows the reached percentage of delivered bundles, the color denotes the routing algorithm, and each sub-figure shows the different payload sizes. For unicasts, successful delivery means that the coordinator receives the bundle, while for broadcasts, all potential recipients (i.e., the civilians) need to receive the bundle. The performance of ProgDTN Emergency is at least equal to all other routing algorithms. In many cases, it outperforms the other routing algorithms, with a delivery ratio of 99.9% in all scenarios. Not even Epidemic Routing (blue) achieves this level of success for high load scenarios, because it produces the highest load and can easily overload the network. Furthermore, PRoPHET (orange) is ill-suited for this scenario and achieves only mediocre results (depending on the experiment, between 43% and 70%). This does not mean that PRoPHET is a bad routing algorithm, but if a scenario does not conform to its assumptions, it will fail to achieve its intended result.

Delivery Times.

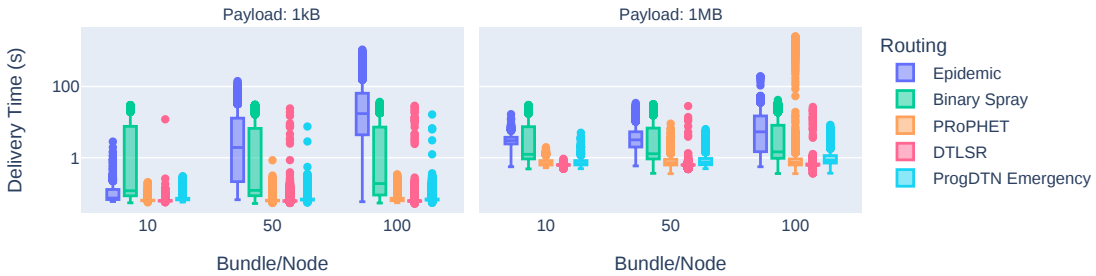


Figure 5.31: Time to deliver a bundle to its destination for different parameters

The next metric is *delivery time*, i.e., the time it takes for a bundle to reach its intended recipient. For broadcasts, we consider the delivery time to the first eligible recipient. The results are shown

in Fig. 5.31; the x-axis shows bundles per node, the y-axis the delivery time on a logarithmic scale, the color denotes the routing algorithm, and each sub-figure shows the different payload sizes. ProgDTN Emergency shows results at least on-par with the other algorithms, with a median below 1 second, regardless of the scenario, and rare cases where the delivery time exceeds 15 seconds for high load scenarios. Epidemic Routing loses performance in higher-load scenarios due to excessive network load. In extreme cases, a bundle can take up to 15 minutes to reach its destination. However, the decrease in long-time outliers for the higher-payload scenario is because we only see initial, fast deliveries for this scenario, while once the system reaches congestion, bundles do not arrive at all and are thus absent from this graph. This observation is also consistent with the observation made for the delivery ratios. Under certain conditions, PRoPHET behaves quite erratic (one hour delivery time), because delivery probabilities are only updated for new connections, which leads to race conditions. This shows that PRoPHET it is not well suited for this scenario. However, all these variations of epidemic routing and PRoPHET are outliers, while the 75% quantile remains quite small, e.g., about 17 seconds for epidemic routing and 1 second for PRoPHET.

Network Load.

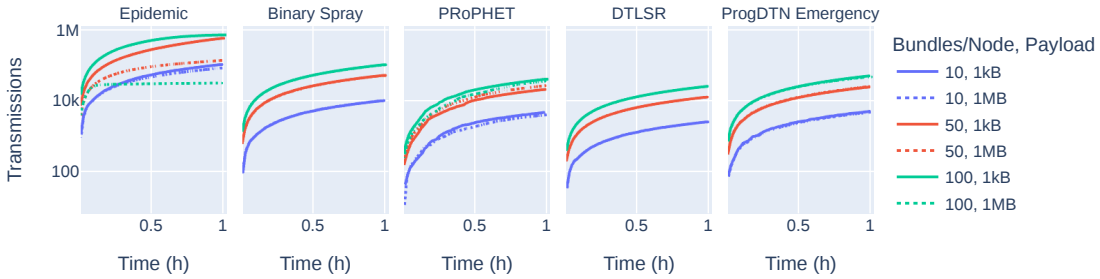


Figure 5.32: Total number of bundle transmissions for different parameters

Fig. 5.32 shows the number of bundle transmissions throughout the experiment, where the x-axis shows the time and the y-axis the transmissions on a logarithmic scale. The color denotes the bundles per node, the line style and payload size, and each sub-plot shows a routing algorithm. From Fig. 5.32 it becomes apparent that Epidemic Routing produces orders of magnitude more transmissions than all other algorithms, i.e., up to 695,000 bundles over one hour compared to about 50,000 bundles for ProgDTN Emergency and 25,000 bundles for DTLSR for 100 bundles per node and a payload size of 1 kB. The cause is the inefficiency of Epidemic Routing; it replicates all bundles to all peers without any concern for whether a transmission increases the delivery probability. Binary Spray & Wait also produces a relatively high load level compared to algorithms other than Epidemic Routing, while the remaining algorithms have a somewhat similar load level. ProgDTN Emergency successfully avoids unnecessary transmissions and conserves network capacity. If this data is viewed in conjunction with Figs. 5.30 and 5.31, it is apparent that ProgDTN Emergency achieves high average delivery ratios of about 99.9% within 1 second, while other routing algorithms suffer in different quality metrics for various reasons.

Routing Decisions.

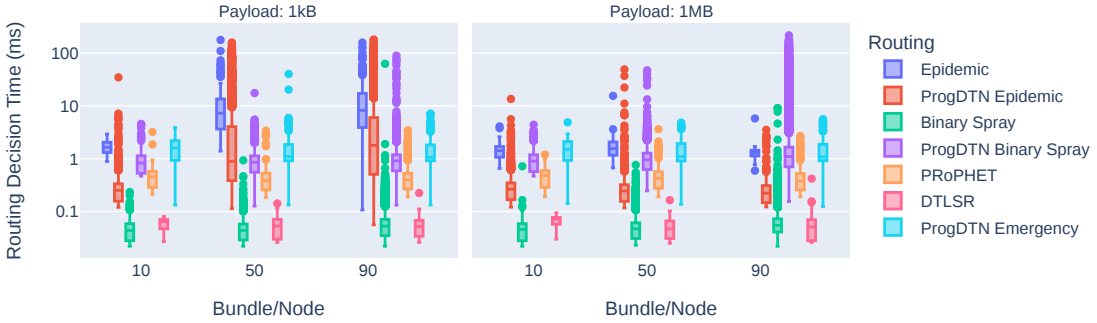


Figure 5.33: Time to make a routing decision for different parameters

Fig. 5.33 shows the time it takes to perform a routing decision for each algorithm. The x-axis groups different bundles per node, and the y-axis shows the time in ms it takes the routing algorithm to finish on a logarithmic scale. Each color represents a routing algorithm, while the two sub-plots show the results for different payload sizes. The focus of this metric is the comparison between Epidemic Routing and ProgDTN Emergency and between Binary Spray & Wait and ProgDTN Binary Spray & Wait, since it shows the overhead introduced by the JavaScript VM. It is evident that the ProgDTN variants of the two algorithms take longer for their routing decisions compared to the non-ProgDTN variants. This is not surprising, since every time a ProgDTN-based algorithm has to make a routing decision, it needs to initialize and start a JavaScript VM and then execute the actual routing code, which is interpreted rather than run as native code. However, the mean and 75% quantile is still well below 50 ms even in those cases. Furthermore, ProgDTN Emergency performs reasonably well with a median of about 1.5 ms and a 75% quantile of about 3 ms, regardless of the experiment. To summarize, the JavaScript VM introduces overhead that is compensated by the fact that ProgDTN can be used to implement a scenario-specific algorithm, reducing the average time to deliver a bundle.

Bundle Overhead.

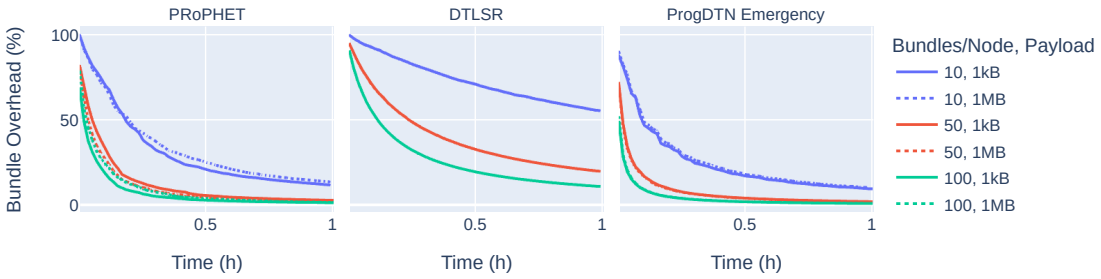


Figure 5.34: Overhead in terms of percentage of bundles sent without a payload

Since ProgDTN makes use of context bundles to let nodes exchange context information (see Section 5.5.3), the amount of additional traffic generated by these bundles needs to be

quantified, since it may contradict the qualitative metrics of preserving network bandwidth. Fig. 5.34 shows the overhead of all algorithms that transmit metadata bundles in terms of the percentage of sent bundles, where the x-axis denotes the experimental runtime, the y-axis the percentage of context bundles (in case of DTLSR so-called meta bundles). The color shows different bundles per node, and the line style represents the two payload sizes. Finally, each subfigure shows a different routing algorithm. Note that only the three shown routing algorithms produce an overhead. Since each node sends a context or meta-bundle upon startup for all algorithms, they all start at 100% overhead; this percentage does, however, decrease rapidly as payload bundles start being transmitted. For both PRoPHET and ProgDTN Emergency, we see an exponential decrease with the overhead percentage converging to zero, but remember that PRoPHET does not achieve a satisfactory delivery ratio in this scenario. DTLSR, on the other hand, regularly broadcasts peer information to the whole network, so we see a higher overhead throughout the experiment.

CPU Usage.

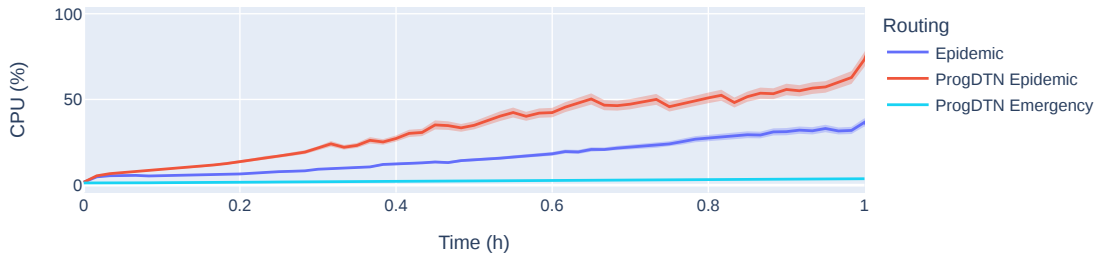


Figure 5.35: CPU usage of three routing algorithms

Fig. 5.35 shows the overhead in terms of CPU usage. The x-axis denotes the experimental time. The y-axis shows the average CPU usage in percent of all nodes. The colors represent different routing algorithms. For this evaluation, we only consider Epidemic Routing, ProgDTN Epidemic, and ProgDTN Emergency to quantify the overhead of the CPU and the potential savings using a custom routing algorithm. The blue curve, representing Epidemic Routing, gives a baseline for system load that correlates with network load shown in Fig. 5.32. Since the network saturation increases over the experimental run, the system load increases in step. The red curve shows the re-implementation of Epidemic Routing in ProgDTN. As expected, it introduces a higher CPU usage, which is due to the increased computational load introduced by the constant re-initialization of the JavaScript VM. The custom routing algorithm, as represented by the turquoise line, by contrast, shows the smallest CPU load, even though it also makes use of the same system as ProgDTN Epidemic. The reason is that if there are fewer bundles to transmit, the JavaScript VM is invoked less often, reducing the overall system load due to fewer transmissions. Thus, by reducing the number of transmissions, we can conserve computing power and therefore also electrical power in cases where a device might be battery-powered.

To summarize, ProgDTN Emergency achieves excellent results and outperforms all other routing algorithms in all metrics. The results of PRoPHET show that a routing algorithm not designed for a particular scenario does not achieve any satisfactory results. Epidemic Routing achieves

good results for smaller payload sizes and a small numbers of bundles per node, but its delivery ratios decrease below 50% for high network load, while ProgDTN Emergency still achieves 99.9%. DTLSR also achieves excellent delivery ratios and delivery times, but its overhead in terms of meta-bundles is significant compared to ProgDTN Emergency. Finally, Binary Spray & Wait performs considerably well, but requires more transmissions to achieve comparable results to ProgDTN Emergency.

5.5.6 Summary

We presented *ProgDTN*, a novel approach to support *Programmable Disruption-tolerant Networking* by allowing network operators to program a node's routing behavior based on context information, without requiring knowledge of the router's interior workings. Our experimental evaluation showed that a programmable DTN routing algorithm tailored to a specific scenario achieves excellent results in terms of up to 99.9% delivery ratio while reducing unnecessary transmissions by 92.9% compared to state-of-the-art DTN routing algorithms in an emergency response scenario. We achieved a low delivery time of bundles (1 – 15 seconds), and low overhead in terms of CPU utilization and routing decisions.

Based on the classification in this thesis, ProgDTN can be considered a smart solution. The classification presented in Figure 5.1 on page 80 compares the approach presented in this section with the conventional routing algorithms DTLSR, PRoPHET, Binary Spray & Wait, and Epidemic Routing in terms of information analysis cost and achievable quality. There are two main QoS metrics: the delivery ratio, i.e., the number of bundles sent that have reached their destination, and the delivery times, i.e., the time it takes for a bundle to reach its intended recipient. The bundle overhead is considered as the primary cost metric, i.e., the percentage of metadata bundles. As additional metrics, local storage, CPU usage, and routing decision time could be compared, but in the experiments it has been shown that the differences are negligible.

	Delivery Rate	QoS Delivery Rate	Median Delivery Time (ms)	QoS Delivery Times	Overall QoS	Bundle Overhead
Epidemic	69.46 %	1.00	3.18	1.00	1.00	0
Binary Spray	90.98 %	1.31	1.30	2.45	1.88	0
PRoPHET	70.77 %	1.02	0.72	4.42	2.72	2.35
DTLSR	96.15 %	1.38	0.62	5.13	3.26	19.94
ProgDTN	99.8 %	1.44	0.75	4.24	2.84	2.11

Table 5.9: Classification of the ProgDTN configuration with 50 bundles per node of 1 MB size

In Table 5.9, bundle delivery rates, delivery times, and bundle overhead as presented in Section 5.5.5 are shown. The overall QoS is the averaged QoS of the delivery rates and delivery times. Regarding information analysis cost, since Epidemic and Binary Spray do not require any additional communication, there is an overhead of 0. As shown in Figure 5.1, while the overall

QoS of DTLSR is higher than the other approaches, the costs are disproportionately high. ProgDTN and PRoPHET are comparable in both QoS and cost, as we can draw from the experimental evaluation that ProgDTN reaches an almost 100% delivery rate compared to about 70% for PRoPHET, while the median delivery time only differs by a few nanoseconds. Depending on the task, the QoS weighting may be different, e.g., for a smart distributed sensing scenario only delivery might be important, while for an emergency response scenario both metrics are equally important. In essence, the approach of ProgDTN allows network operators to trade off information analysis cost for QoS depending on the particular scenario.

5.6 LoRa-based Device-to-Device Smartphone Communication

5.6.1 Introduction

The communication technologies developed and deployed in the last decades are integral parts of our daily life and are used by mobile phones, computers, or smart applications in homes and cities. Today's smartphones, however, highly depend on the availability of telecommunication infrastructures, such as Wi-Fi or cellular technology (e.g., 3G, LTE, or the upcoming 5G standard). However, there are situations in which either no communication infrastructure is available or only at a high cost, e.g., in remote areas ([Gar11]), in the agricultural sector ([Eli+18]), as a result of disasters ([MB07]), or due to political censorship ([Liu+15]). Furthermore, in countries with less evolved infrastructures, e.g., due to low population densities or due to economic reasons, cellular networks often cannot be used at all or cannot be established in an economically feasible manner. In this case, low-cost communication technologies would give people the possibility to communicate with each other ([KW16]). However, while modern infrastructure-independent technologies do exist, these are often only accessible to advanced users due to regulations, high costs, or technical complexity. To make these technologies accessible to a broad user base, they need to be integrated into devices already known to users.

We propose to use LoRa wireless technology as a communication enabler in such situations. LoRa (*Lo*ng-*Ra*nge) is a long range and low power network protocol designed for the Internet of Things to support low data rate applications ([Hor10]). It consists of a proprietary physical layer, using the Chirp Spread Spectrum (CSS) in the freely usable ISM bands at 433, 868, or 915 MHz, depending on the global region. The additional MAC layer protocol LoRaWAN is designed as a hierarchical topology. A set of gateways is receiving and forwarding messages of end devices to a central server that processes the data. While LoRa itself has to be licensed by the Semtech company and implemented in specific hardware, it is independent of LoRaWAN and can thus be used in a device-to-device manner.

In this section, an approach to equip existing mobile devices with LoRa technology, by distributing small System-on-a-Chip (SoC) devices supporting multiple Radio Access Technologies (RATs), is presented. There are several commercially off-the-shelf microcontroller units (MCUs) available supporting Wi-Fi, Bluetooth, and LoRa. We propose to use these low-cost devices to upgrade existing smartphones, laptops, and other mobile devices for long range infrastructure-less communication. To reach this goal, we present a custom firmware for Arduino-SDK compatible boards, called *rf95modem*. Existing mobile devices can be connected to a board through a serial connection, Wi-Fi, or Bluetooth. As a general solution, we propose to use modem AT commands

as an interface for application software. This interface can then be exposed through different communication channels and used by application software without requiring LoRa specific device drivers. Since these boards are cheap and do not require laying new cables or setting up communication towers, these boards can either be distributed to people living in high-risk areas beforehand or handed out by first responders during the event of a crisis.

To demonstrate the functionality of our implementation, we first present a cross-platform mobile application for device-to-device messaging. This re-enables basic infrastructure-less communication capabilities in disasters. Second, we present an integration of our implementation into a disruption-tolerant networking (DTN) software. Although the low data rates of LoRa are not sufficient to support multimedia applications, sensor data, e.g., in agricultural applications or environmental monitoring, as well as context information for further DTN routing decisions can be transmitted through the LoRa channel. To illustrate the benefits of our approach, the developed device-to-device messaging app, as well as our DTN integration are tested through experimental evaluations in an urban and a rural area.

To summarize, we make the following contributions:

- We present a novel free and open source modem firmware implementation for LoRa-enabled MCUs, featuring a device-driver independent way of using LoRa via serial, Bluetooth LE, and Wi-Fi interfaces.
- We present a novel device-to-device LoRa chat application for a) Android and iOS smartphones and b) traditional computers.
- We present a freely available and open source integration of long range communication into a delay-tolerant networking software.
- We experimentally evaluate the proposed approach by conducting field tests in an urban environment as well as in a rural area and performing energy measurements of multiple devices.
- The presented *rf95modem* software[42], the device-to-device chat application[43], the integration into DTN7[44] and the experimental evaluation code fragments[45] are freely available.

Parts of this section have been published in Jonas Höchst, Lars Baumgärtner, Franz Kuntke, Alvar Penning, Artur Sterz, and Bernd Freisleben. “LoRa-based Device-to-Device Smartphone Communication for Crisis Scenarios.” in: *17th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2020)*. Blacksburg, Virginia, USA, May 2020.

Parts of this section will appear in Jonas Höchst, Lars Baumgärtner, Franz Kuntke, Alvar Penning, Artur Sterz, Markus Sommer, and Bernd Freisleben. “Mobile Device-to-Device Communication for Crisis Scenarios Using Low-cost LoRa Modems.” in: *Disaster Management and Information Technology: Professional Response and Recovery Management in the Age of Disasters*. ed. by Hans Jochen Scholl, Eric E. Holdeman, and F. Kees Boersma. Springer Nature, 2022.

[42] `https://github.com/gh0st42/rf95modem/`, MIT License
[43] `https://github.com/umr-ds/BlueRa`, MIT License
[44] `https://github.com/dtn7/dtn7-go`, GNU General Public License v3.0
[45] `https://github.com/umr-ds/hoechst2020lora`

5.6.2 Related Work

[Aug+16] experimentally evaluated the foundations of LoRa. The authors built a LoRa testbed and conducted different tests including receiver sensitivity and network coverage. LoRa's Chirp Spread Spectrum (CSS) modulation technique allows to decode received signals from -120 to -125 dBm, depending on the spreading factor (SF). The network coverage was examined in a suburb of Paris using SFs of 7, 9, and 12, based on different test locations. With SF7 and SF9, distances of 2.3 km were reached with less than 50% packet loss. Using SF12, the packet delivery ratio at the highest distance of 3.4 km was 38%.

[BVR16] investigated the current LoRaWAN protocol and proposed an alternative MAC layer to be used with LoRa, making use of multi-hop communication. [Wix+16] evaluated the properties of LoRaWAN for wireless sensor networks, demonstrating reliable usage of LoRa up to 2.2 km in an urban scenario.

[Bau+18] proposed to use LoRa for environmental monitoring. In the included LoRa evaluation, ranges of 4.6 to 6.5 km with the base station placed on a high building were achieved depending on the antenna and the frequencies in use. Furthermore, the concept of a unified radio firmware was introduced, but only limited functionality was implemented and evaluated.

Long range peer-to-peer links were investigated by [Cal+19]. The authors showed experimentally that with an increased SF the Received Signal Strength (RSS) did not change but the Signal to Noise Ratio (SNR) was increased, proving the better decoding ability. Distances of up to 4 km in a line-of-sight and 1 km in a forested terrain were achieved.

[Dee+19] created an overview of wireless technologies for post-disaster emergency communication. They identified three disaster network scenarios: congested network, partial network, and isolated network. In isolated networks, the user devices have to deploy a new network to provide temporal wireless coverage. This could be achieved with drone-assisted communication or mobile ad-hoc networks (MANETs). The advantage of the latter is high redundancy: a failure of individual nodes is not necessarily mission-critical.

[Lie+17] analyzed multiple disaster scenarios to highlight the main communication issues that occurred. The depicted scenarios are based on unavailable or broken communication infrastructures. In particular, the authors proposed an architecture that incorporates delay-tolerant MANETs to be independent of any fixed infrastructure. Additionally, the authors focused on communication tools that ordinary civilians can use, since civilians typically do not possess their own dedicated communication facilities, in contrast to disaster relief organizations.

By analyzing 49 crisis technology articles that focus on mobile apps in disaster situations, [Tan+17] illustrated that disaster communication is shifting away from authority-centric approaches towards approaches that integrate and engage the public. The authors argued that supporting on-site collaboration (e.g., by chatting) is the main purpose of mobile apps for disaster situations.

According to [Kau+18], the widespread use of smartphones provides opportunities for bidirectional communication between authorities and citizens. The authors developed the app

112.social for communication between authorities and citizens during emergencies. The authors argued that further research in the area of infrastructure-less technologies for emergency communication apps is required to provide new opportunities.

[Sci+18] presented an infrastructure-less solution for emergency communication by combining LoRa modules with smartphones. In their approach, the LoRa transceiver was hooked directly to the smartphone via USB to achieve higher communication ranges compared to conventional wireless transmission technologies (e.g., Wi-Fi). Thus, only Android devices work with this approach, and the solution is tightly coupled to the emergency communication app provided by the authors. [Olt+13] used an USB dongle to access ZigBee nodes through an Android app. These USB connected devices were later also identified by [STD20] as being problematic and tackled through the addition of an extra Bluetooth bridge. This setup is still tailored to the provided emergency application of the authors and has higher complexity, bill of materials, and energy consumption compared to our approach.

5.6.3 Design

In this section, the design goals of the proposed approach are discussed. First, general principles of using LoRa on smartphones are covered. Second, design goals of a generic LoRa modem firmware are presented. Third, requirements for a device-to-device chat application are examined. Finally, thoughts on integrating LoRa into disruption-tolerant networking are presented.

Enabling LoRa on Smartphones

To extend smartphones and other common devices by infrastructure-less communication technologies, a generic interface must be designed. While these devices offer a variety of communication technologies, only few are shared across different categories of devices. Ethernet and USB may be available on most devices including laptops and routers, but smartphones can utilize these connections only using special adapters, if at all. However, all of the mentioned devices offer Wi-Fi and/or Bluetooth interfaces. Furthermore, the used approach should be based on low-energy solutions, since in the described scenarios power supply may be limited or not available. In the following, we present a modem firmware, called *rf95modem*, for LoRa MCUs that can enable access to the LoRa hardware through other communication channels.

Modem Firmware

Figure 5.36 shows how different devices can be connected to a modem board. There are several commercial off-the-shelf micro-controller boards available that include a LoRa transceiver and thus can be used for the proposed functionality. With our approach, we aim to support the majority of these boards by providing a hardware abstraction layer across all of them. Thus, the provided implementation supports a wide variety of available boards, e.g., the LilyGO TTGO

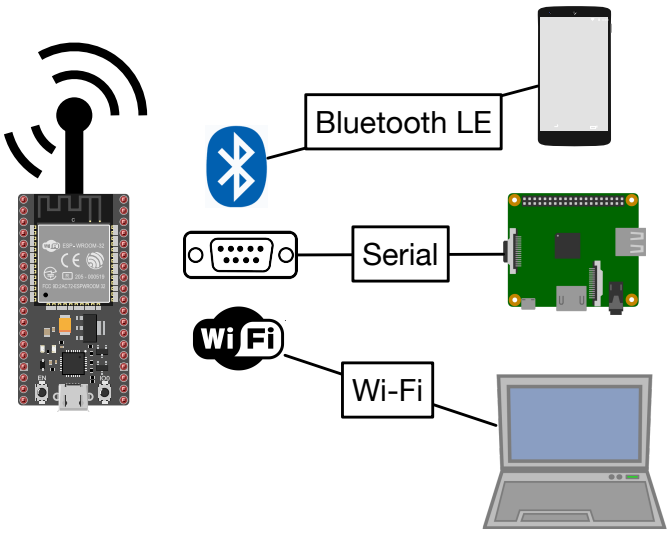


Figure 5.36: ESP32-based modem board and its connection options for smartphones, single-board computers, and laptops.

LoRa series[46], Adafruit's Feather 32u4 and M0 boards [47] or the Heltec Automation WiFi LoRa 32 and Wireless Stick (Lite)[48]. Some of these boards only provide LoRa and a serial interface via USB, but others also provide Wi-Fi and Bluetooth. The modem firmware is supposed to be controllable through AT commands similar to classic modems or various smartphones. Thus, no specific device drivers are needed to send and receive data via the *rf95modem* firmware. Finally, the firmware should be flexible enough and easily configurable to only ship the code actually needed for the device and the scenario in which it is used.

Incentivizing LoRa Usage

An important challenge in establishing emergency networks is the availability of hardware and software to users when an emergency happens. If users find themselves in an emergency situation with an infrastructure failure, they either have to wait until the infrastructure is restored, they are equipped with new technology, e.g., from the emergency services, or they can use existing and already known devices and technologies. The latter case has the clear advantage that rudimentary communication and, in particular, emergency calls are possible without involving third parties. For the technology presented here to have an impact, it is necessary for users to be able to use it meaningfully outside of a crisis, to gain experience with it, and to avoid the need for elaborate steps in the event of a crisis. In this section, we outline some use cases where LoRa-based communication is helpful in everyday life and can get users to familiarize themselves with LoRa technology.

[46] `http://www.lilygo.cn/pro.aspx?FId=t3:50003:3`, Xing Yuan Electronic Technology Co., Ltd., LongGang, Shenzhen, China

[47] `https://www.adafruit.com/product/3178`, Adafruit Industries, LLC, 150 Varick Street, New York 10013, USA

[48] `https://heltec.org/proudct_center/lora/lora-node/`, Heltec Automation, Longtan Industrial Park, Chengdu, China

A strong use case for LoRa outside of emergency communication are outdoor activities, in which people are in areas of bad or completely missing cellular coverage. While in skiing areas cellular networks are built due to commercial interest, many activities depending on less infrastructure suffer from a missing communications infrastructure. Using LoRa communication among the participants of a group or even among different groups in the same area can be very useful for coordination, e.g., if a part of the group separates and looks for food, water or firewood. Even in the case of unintentional separation, e.g., if the group gets lost while canoeing, this infrastructure-free communication can be helpful to find each other again. There are several commercially available products that support the case for a companion device offering infrastructure-free communication, such as GoTenna[49], MeshTastic[50], or Beartooth[51].

A second incentive for using LoRa is gamification, e.g., by deploying beacons through volunteers in certain locations. Beaconing can be implemented as a service on already existing LoRa infrastructure, such as on LoRa gateways or even on weather stations or other IoT devices. The goal of the game would be to collect as many beacons as possible and thus prove that a player has actually visited the locations. To make cheating in the game more difficult and to introduce another component for the competition of different players, the beacons are generated and signed based on a timestamp.

A third use case for LoRa in everyday life is a public message board that is enhanced by local information. Important information of the city, e.g., for visitors but also for people who live in the city can be announced via LoRa, e.g., local weather recordings, traffic information, or information in potentially dangerous situations, such as power outages, fires or terrorist attacks. The inherent property of a location-based limitation allows effective and efficient distribution of location-based information and can also be used for marketing purposes.

A Device-To-Device Messaging Application

To enable communications in rural areas or in situations after disasters, mobile applications play an outstanding role for various reasons and support a variety of communication technologies like cellular, Wi-Fi, and Bluetooth. Therefore, we designed a mobile application to support off-grid communication in the scenarios mentioned above. In particular, in crisis situations, it is important that users do not first have to familiarize themselves with new paradigms or UI/UX concepts and are not confronted with technical terms that are incomprehensible to laypersons. Therefore, our application should use Bluetooth Low Energy (BLE) as the primary connection technology. Bluetooth is widely accepted as a technology to create one-to-one connections and exchange data between the involved peers, whereas Wi-Fi is usually used to access information from a central place. Therefore, the Bluetooth paradigm fits better to the given scenario. Additionally, Bluetooth is more energy efficient compared to Wi-Fi, which makes it the appropriate technology to use in this case. To further reduce barriers in app usage, our app should automatically connect to nearby modem devices without any further actions required by the user. This increases the chances of instant access to the communication infrastructure in cases of emergencies. The app should also be able to receive messages in the

[49] `https://gotenna.com`
[50] `https://meshtastic.org`
[51] `https://beartooth.com`

background, e.g., when the user leaves the application. Additionally, the user should not worry about using a specific mobile device. It is therefore crucial to provide a platform-independent application that is usable on the most popular mobile platforms iOS and Android.

To get people in contact as fast as possible without prior exchange of IDs, usernames or alike, the application should follow a public message board paradigm, similar to Twitter, where users can post short messages to a publicly visible channel. Here, users can send messages visible for all and ask for help or provide status information. This approach gives users easy and fast access to a communications method. Users should have an easy and fast way to find new channels and create channels for specific topics. Finally, users should be presented with a common and familiar look-and-feel including accessibility features so that no one is excluded.

Disruption-tolerant Networking

For crisis scenarios, DTN is a technology to enable infrastructureless communication using an emergency infrastructure in conjunction with existing devices of users ([Bau+16; Lie+17]). DTNs benefit from a large number of devices storing and forwarding messages to other devices when they become available. Today, end user focused DTNs are mostly based on ad-hoc Wi-Fi and Bluetooth, since these are available in the mobile devices used by the users. Due to slow data rates and duty cycle restrictions introduced by regulation, LoRa in DTNs is not suited for larger data transmissions, such as multimedia content, but is helpful to transmit context information or small messages. LoRa can be used to connect different local clouds of people, where smaller messages available inside the cloud can be transmitted to another cloud. Modern DTN routing algorithms use context information to reduce overheads introduced by unnecessary transmission ([Gra+18a]). We propose to add LoRa to existing delay- and disruption-tolerant networks to enable larger spatial low-bandwidth coverage, in order to propagate small messages and context information. To facilitate the use of LoRa in DTN networks, an exemplary integration should be implemented that can use LoRa via Bluetooth, Wi-Fi, or a serial connection and thus is available on mobile devices and static nodes added in crisis scenarios.

5.6.4 Implementation

In this section, the implementations of the *rf95modem* firmware, the device-to-device messaging application, and the integration into disruption-tolerant networking software are presented.

Modem Firmware

Since a *rf95modem* should be controlled by AT commands over all of its available connection mechanisms, handling such commands is an essential part of the implementation. Therefore, this functionality is shared across all supported hardware platforms and connection mechanisms, as shown in Figure 5.37. Here, the software components are displayed in blue while the rest represents the underlying hardware modules. For interaction with users and software, the serial device interface, usually accessible via USB, is always active. Furthermore, the in-/output

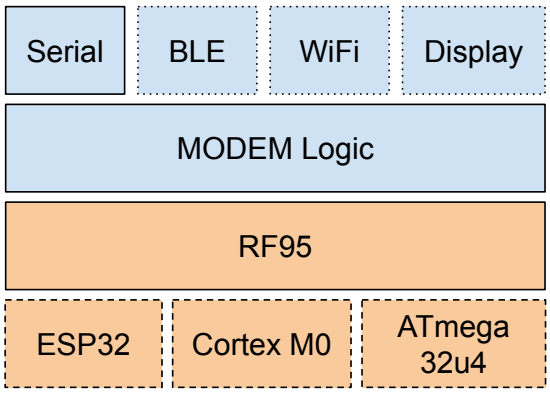


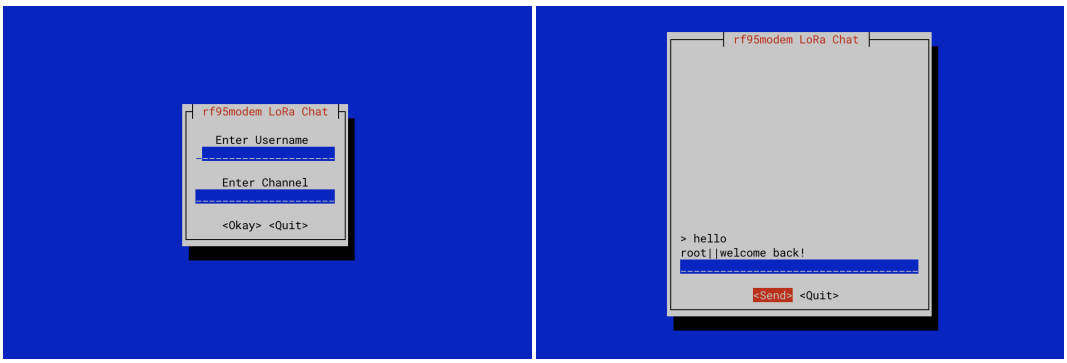
Figure 5.37: Overview of the *rf95modem* architecture.

functions may also be hooked to the Bluetooth Low Energy or WiFi modules we developed, if enabled at compile time. Any output is mirrored to all enabled interfaces that can be used simultaneously.

To achieve optimal results on various hardware platforms and configurations, all features and hardware configurations can be set using build flags. For example, the SPI pin configuration and the underlying CPU architecture must be configured, as well as the base LoRa frequency. Currently, we support ESP32-based boards with RF95-compatible LoRa transceivers as well as some Cortex M0 and ATmega32u4-based boards, such as the ones produced by Adafruit for the Feather line of devices.

For the ESP32 boards, we provide a WiFi mode featuring two different ways of communication that can be used in parallel. In both cases, an access point is opened by the device itself for modem users to connect to. The first mode is UDP-based and just broadcasts the modem output to the local network and interprets incoming AT commands via datagram packets. This is especially useful if many local devices want to listen on incoming transmissions. The second mode is the TCP exclusive mode. Here, a single TCP connection is accepted that can then control the model similarly to a serial interface. Since the ESP32 boards also feature Bluetooth, they can be used to announce a BLE characteristic for interaction with the *rf95modem*. This interface acts similarly to the others by interpreting strings received via a write characteristic as AT modem commands. The output is shared via a notify characteristic to which devices can subscribe. BLE is supposed to have a payload limit of 20 bytes, and thus splitting the serial output into smaller chunks is necessary. Our tests on various platforms, e.g., iPhones and Raspberry Pis, have shown that sending much larger packets via BLE is often also possible and much more efficient. Therefore, sending overlong frames via BLE can optionally be activated during runtime via a specific AT command. Finally, there is also a software module to support OLED displays as they are pretty common on TTGOs and Heltecs ESP32 devices. If enabled at compile time, these can be used to display status information such as the current frequency, packets received, and number of packets transmitted, which can be used for debugging or providing statistical information at a glance without the need for special hardware or software.

Since all board-specific features can be configured at compile time, the firmware can be custom-tailored to fit even very resource-limited devices. Enabling all features at once results in a large



a) Login screen b) Chat interface

Figure 5.38: Console-based *rf95modem* LoRa chat example.

firmware, which requires more flash memory and a custom partition layout, but still works on the most common ESP32 boards. Due to the fact that all output is mirrored between the interfaces, one can easily use two interfaces in parallel, e.g., debugging the BLE communication via an attached serial cable. The firmware is completely written in C/C++ using the Arduino SDK and PlatformIO as a build system.

A Device-To-Device Messaging Application

To satisfy the requirements of the messaging application, we provide two different approaches. First, we provide a console-based user interface for traditional computers, as shown in Figure 5.38[52]. Second, for the mobile version of the application (BlueRa), we used the Flutter UI toolkit[53]. Flutter allows developers to create platform independent apps for both major mobile operating systems, iOS and Android, using the same code base.

Figure 5.39 gives a simplified overview of the components of the app. The top block shows the UI classes. The application starts at the home screen, which contains a path to the settings, a list view of the available channels and a path for joining to new channels or to create channels. On the left, users can change their usernames or manage the app's Bluetooth connection, each in their own screens (the username settings screen is not shown in the figure). When the app's route heads over to the *JoinChannelScreen*, a list of available channels that the user has not joined yet is presented. Additionally, this screen enables the user to create new channels. The final screen is the chat screen itself, where the user can see a history of the messages in this particular channel as well as a text field for creating and sending new messages.

Figure 5.40 shows the chat screen for the announcements channel. Using this common chat UI/UX, the user gets a familiar look and can start messaging immediately, without the need of getting familiar with a special UI.

As indicated by the *Channel* module in Figure 5.39, a channel has a name, an indicator whether the local user has joined this channel and a list of messages. A message, on the other hand,

[52] `https://github.com/gh0st42/rf95modem-rs`
[53] `https://flutter.dev`

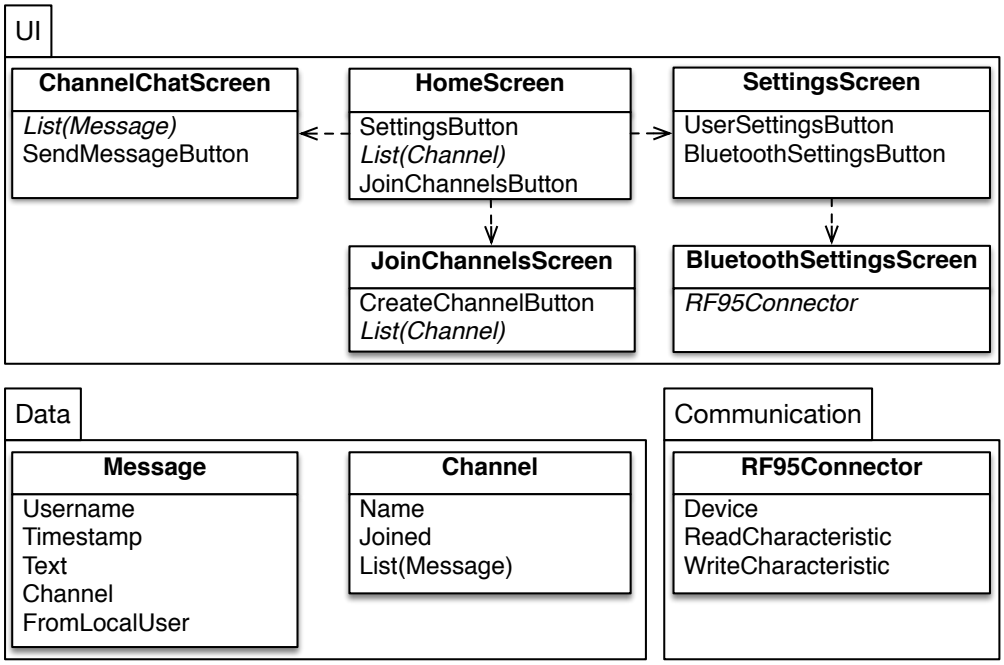


Figure 5.39: Overview of the components of the app.

contains the name of the user who sent this message, a timestamp, the text itself, the channel name and an indicator whether the message was sent from the local user.

The connection to the *rf95modem* device is implemented in its own module, *RF95Connector*. This module holds the device ID and Bluetooth connection state, as well as the read and write characteristics for the serial communication service. Additionally, this module also implements data and message handling. When sending a new message, all required data is serialized to the appropriate format and sent to the modem using the write characteristic. Furthermore, a receive listener gets notified, as soon as new data is available in the read characteristic. The received data is parsed and the internal channel- and message database is updated. If the channel of the received message is already present, the message is appended to the channel's message list. Otherwise, a new channel in the local database is created with the received message. This new channel will be presented in the *JoinChannelScreen*, so that users can join this channel if they want to.

We used a simple communication protocol for sending and receiving messages. It consists of the channel name, a user name, an optional location and the actual message separated by vertical bars. Following this simple protocol design, it is possible for any communication device to communicate with the app, regardless of the capabilities and available serialization libraries like JSON, CBOR, or Protocol Buffers.

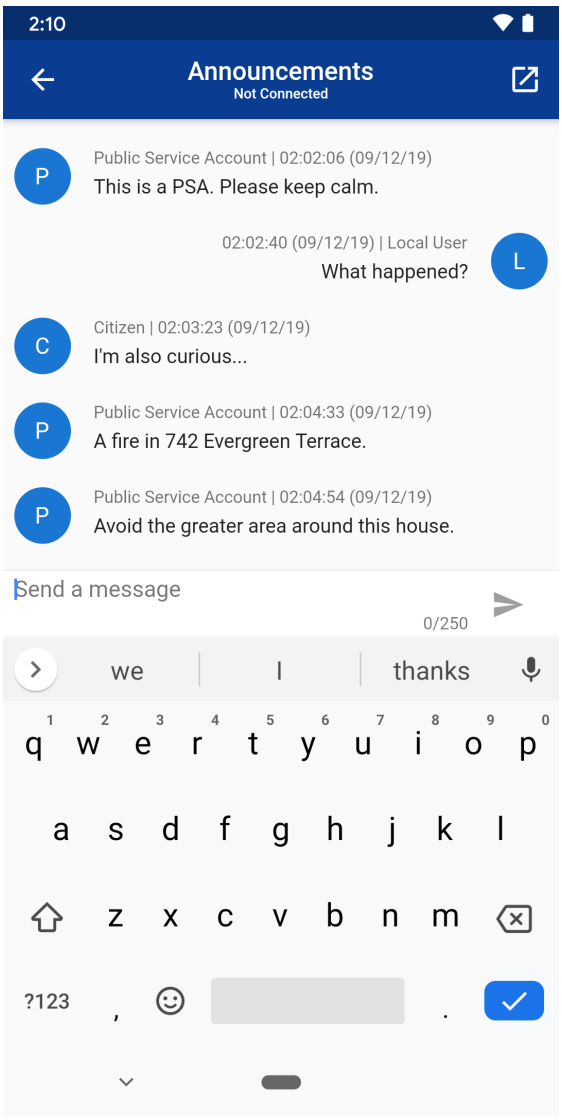


Figure 5.40: Screenshot of the chat screen for the announcements channel.

Disruption-tolerant Networking

To use LoRa in disruption-tolerant networking, we have extended the DTN7 implementation[54] introduced by [Pen+19]. Within the DTN context, the communication interface for bundle exchange between nodes is called convergence layer. We have implemented the convergence layer interface provided by DTN7 to achieve LoRa support.

To integrate *rf95modem*'s serial link into DTN7, we have first developed a library[55], written in the Go programming language. This library's main task is to provide Golang typical interfaces

[54] `https://github.com/dtn7/dtn7-go`
[55] `https://github.com/dtn7/rf95modem-go`

for writing and reading data streams through *rf95modem*. Furthermore, status information of the modem can be read and reconfigured.

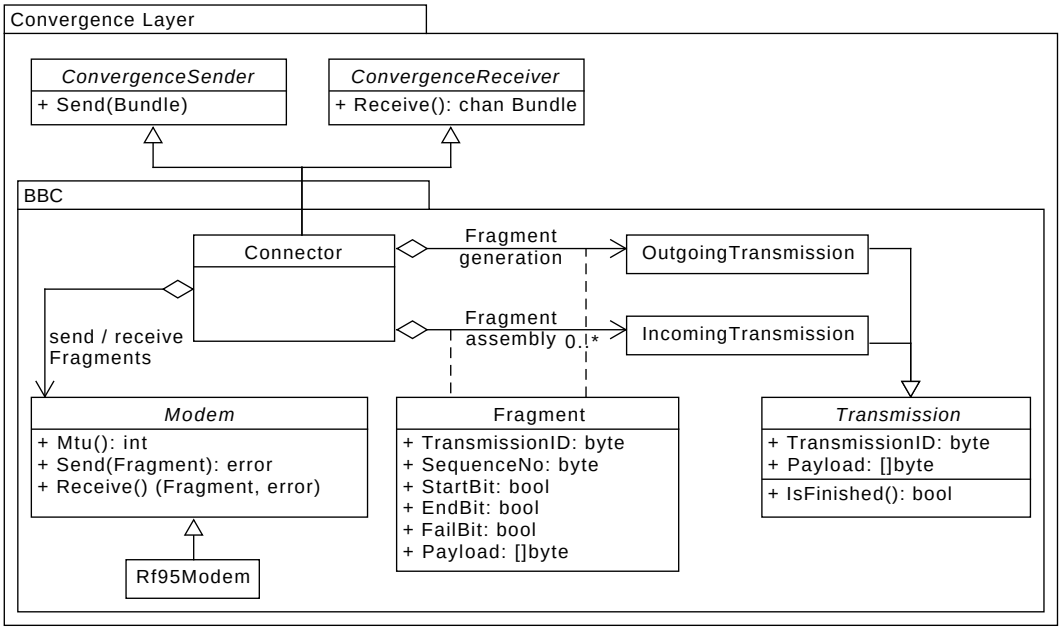


Figure 5.41: Simplified implementation model of the Bundle Broadcasting Connector.

Until now, DTN7 only had support for unicast convergence layers, while the transmission of LoRa packets corresponds to a broadcast. Since most broadcast technologies are similar in structure, we first developed a generic broadcasting convergence layer, the Bundle Broadcasting Connector (BBC). Its simplified implementation model is shown in Figure 5.41.

The main component of the BBC package is the connector that implements DTN7's convergence layer interfaces for both sending and receiving bundles. The connector itself communicates with a modem, which is an interface implemented in *rf95modem-go* and a mock object for testing. Each modem reports its MTU such that transmissions can be fragmented accordingly.

With regard to transmissions, the BBC makes a distinction between incoming and outgoing ones. Both types have an identifier and can determine whether they have finished. If a bundle should be sent via our BBC, an outgoing transmission with a new identifier will be generated. This identifier is derived from the node. Every node is initialized with a random identifier, which is then incremented for each transmission. The payload is the `xz`[56] compressed bundle. As long as the transfer is not completed, the connector requests a new fragment. Its length including headers must not exceed the modem's MTU. This is then handed to the modem, which broadcasts it via LoRa in our case.

The network protocol specification of a fragment is shown in Figure 5.42. A fragment itself consist of a header of two bytes, followed by the payload. In the header, the identifier of the transmission is referenced next to a sequence number. Each fragment contains the incremented

[56] `https://tukaani.org/xz/format.html`

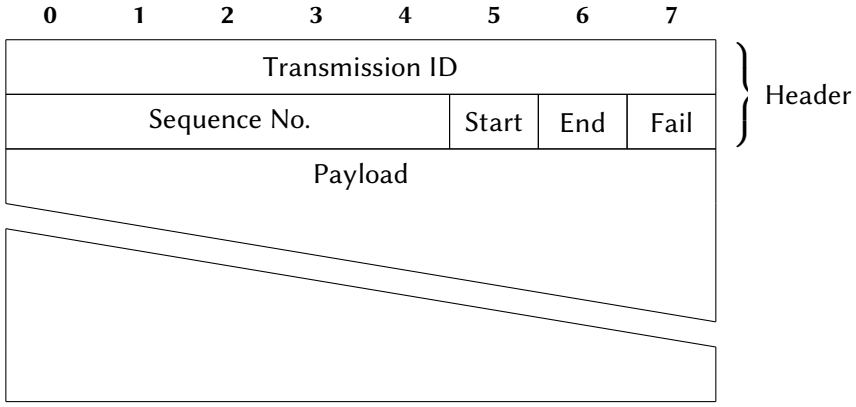


Figure 5.42: Protocol specification of a fragment.

sequence number of its predecessor. Thus, lost fragments can be detected in advance. In addition, the header has three flags. The start bit indicates the beginning of a new transmission, while the end bit indicates its end. A fail bit is set for status packets that imply the absence of fragments.

When receiving fragments, the modem forwards them to the connector. This checks whether the transmission identifier is already known. If this is the case, the fragment is added to the incoming transmission. Otherwise, a new incoming transmission is created. Once the transmission is finished, the entire payload is extracted and decompressed. The resulting bundle will be passed back to DTN7's logic. However, if a reception error occurred, e.g., due to a skipped sequence number, a status packet is sent. This packet is equal to the last fragment, except that the fail bit is set and the payload is empty. Reception of such a packet by the sender marks the transmission as faulty. As a result, DTN7 will re-trigger the transmission at a later time.

5.6.5 Experimental Evaluation

In this section, LoRa protocol properties are discussed, and the presented implementations are evaluated through experiments.

LoRa in Device-to-Device Scenarios

LoRa as a long range protocol is limited in terms of bandwidth, since the resilient encoding scheme introduces some overhead and a duty cycle needs to be followed to fairly use the shared medium. To understand the limitations of LoRa communication, some application-oriented examples are discussed.

Figure 5.43 shows the payload sizes compared to the airtime required for sending with different spreading factors (SF), where the coding rate is set to 4/5. The presented SF and channel bandwidth examples are taken from the EU standards ([All18]). The message length of LoRa is limited depending on the SF to limit the airtime each individual message requires. The highest SFs are limited to a payload of 51 bytes. Using SF9, the payload can can go up to 115

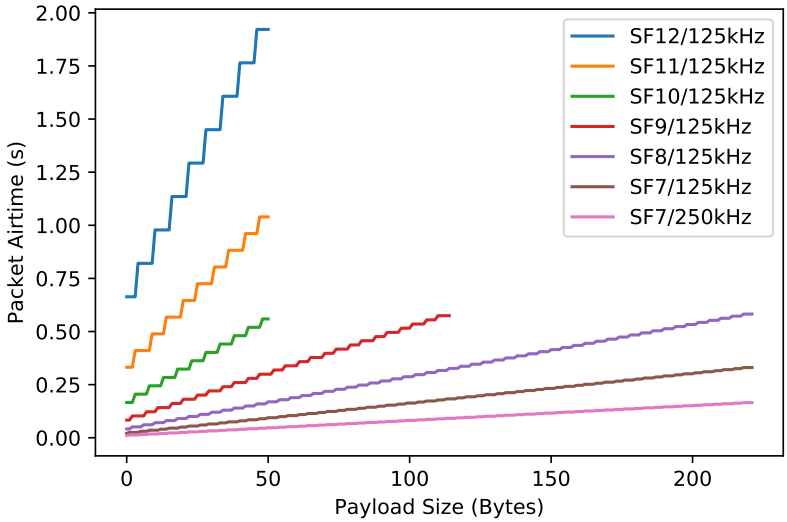


Figure 5.43: Exemplary packet airtime in different LoRa profiles.

bytes, and in the fastest SFs 8 and 7, messages can contain up to 222 bytes. SF12 packets, with the maximum payload of 51 bytes, take up to 1.92 seconds airtime, while 222 bytes in SF7/250 kHz only take 0.16 seconds. When using LoRa for emergency communication, different profiles can be used to model, e.g., the importance of messages. Public service announcement of governmental institutions, including messages of rescuers can be sent in more resilient configurations, while chats of users helping each other in emergency situations can be limited to smaller areas, to cope with the limitations of the protocol.

Device-To-Device Smartphone Communication

We evaluated our proposed infrastructure-less LoRa communication via real world tests that cover two scenarios: (a) city area communication, and (b) rural area communication.

The motivation for scenario (a) are communication demands in disaster situations. By having a low-cost companion device that extends the infrastructure-less communication range of our everyday devices could be a real benefit for such scenarios. However, the inherent characteristics of cities, e.g., the high density of buildings, are a major problem for each wireless technology.

Scenario (b) is motivated by the fact that some rural areas, also in industrial countries, are still not covered by mobile networks (GSM, 3G, 4G, 5G). The expectations of the tests in the rural areas therefore differ, since regions without obstacles might easily get good coverage, while areas with many trees might suffer from worse connections.

Experimental Setup

For the conducted tests, we used one fixed and one mobile station. The fixed station consists of a laptop logging the incoming messages. Figure 5.44 shows the mobile station, consisting of a smartphone in combination with a Heltec Wireless Stick driven by a Powerbank. The default antenna was replaced by a +3dBi model, connected via SubMiniature version A (SMA). The antennas of each station were 1.5 meter above the ground, in order to model realistic usage in device-to-device scenarios.

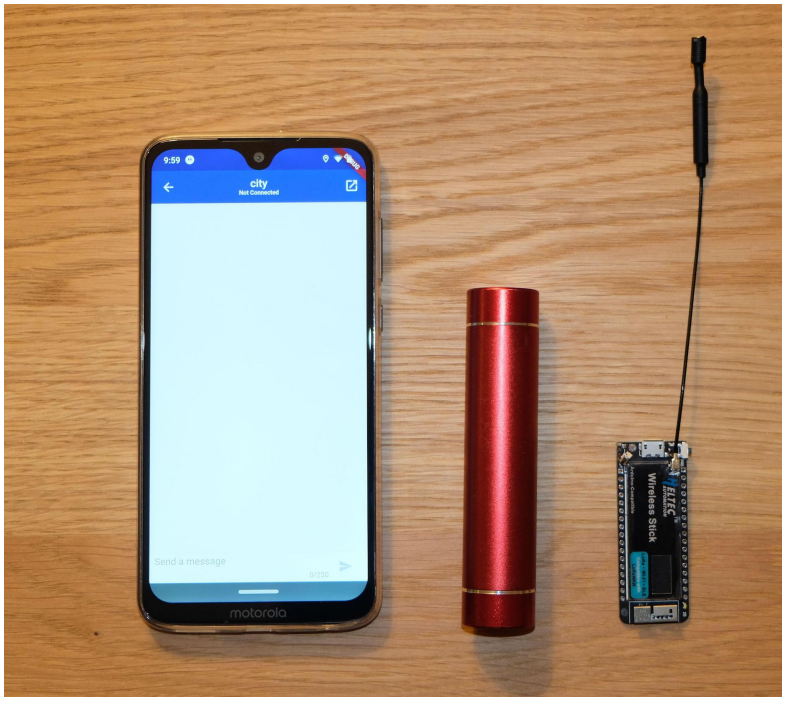


Figure 5.44: Mobile station: smartphone, power bank, and Heltec wireless stick.

First, we selected one exemplary region for each of our two considered scenarios. The fixed station was then placed in the middle of the selected area and started listening for incoming messages. For reproducibility and accuracy, we scripted message generation and sending on the mobile station, such that every 15 seconds one message including a GPS position was sent via Bluetooth LE and broadcasted by the companion device. The mobile station was then moved away from the static station until no message could reach its counterpart anymore. To observe a realistic model of device-to-device communication, the mobile station was moved in multiple directions. The tests in both scenarios were repeated using two LoRa profiles provided by *rf95modem*: (a) Medium Range: Bandwidth: 125 kHz, Cr: 4/5, SF7, and (b) Long Range: Bandwidth: 125 kHz, Cr: 4/8, SF12. Due to the simplicity of our test procedure, we did not get the maximum possible distances of our exemplary regions, but two real world setups, with distances that work even with the simple out-of-box experience of the rather low-cost Heltec wireless sticks.

Results

By analyzing the logs of the smartphone applications that transmit GPS locations of each sent message, we were able to calculate the distances of reliable communication setups between all participants for each scenario.

Scenario	Mode	Maximum Distance
City	(a) Medium Range	1.09 km
	(b) Long Range	2.89 km
Rural Area	(a) Medium Range	1.31 km
	(b) Long Range	1.64 km

Table 5.10: Maximum distances achieved in the different areas and tested LoRa profiles in the conducted experiments

Table 5.10 shows the maximum distances of the conducted tests. For the Medium Range configuration, 1.09 km in the city area and 1.31 km in the rural area could be achieved. With the rather high data rate of 5.47 kbps, the mode is a good choice in dense areas, where a larger amount of messages might occur, and airtime is limited. In the Long Range profile, 1.64 km could be achieved in the rural area, while in the city scenario, some messages could be transmitted from 2.89 km range.

Figure 5.45 shows the results of the conducted tests in the city area. The orange dots denote the Medium Range profile, while the red dots show the successful transmissions in the Long Range profile. In the size of the markers, the Received Signal Strength Indicator (RSSI) is visualized. The larger the marker, the better the RSSI is. Note that in LoRa a higher SF enables a higher chance of successful decoding under worse RSSI values. In the presented results, it is evident that LoRa works well as long as no obstacles are in the way. The maximum distance in the city area was achieved in the valley going through the city. Even though obstacles, such as buildings, were in the way, the signal could reach the other peer well. When moving behind a hill, such as in the western or eastern parts of the presented map, the signal was not able to penetrate the obstacle.

In Figure 5.46, the successful LoRa transmission of the rural area are presented. As excepted, the transmission range in the forested area is worse compared to the unforested area. In the presented example, the northern part of the map consists of a forested area, while the southern part is mostly not forested. From the plot, it can be observed that in the non-forested valley area, RSSI is high, and all LoRa messages are successfully transmitted in both modes. When forested areas and hills are in the line of sight, the RSSI worsens and quickly becomes unavailable. In Long Range mode, transmission in forested places improves and messages are successfully transmitted through up to 600 meters of forested area.

In Figure 5.47, the observed RSSI values in relation to the distances are presented. With the Long Range profile, signals with RSSI values of up to -140 dBm can be decoded successfully, while in the Medium Range profile the limit is around -130 dBm.

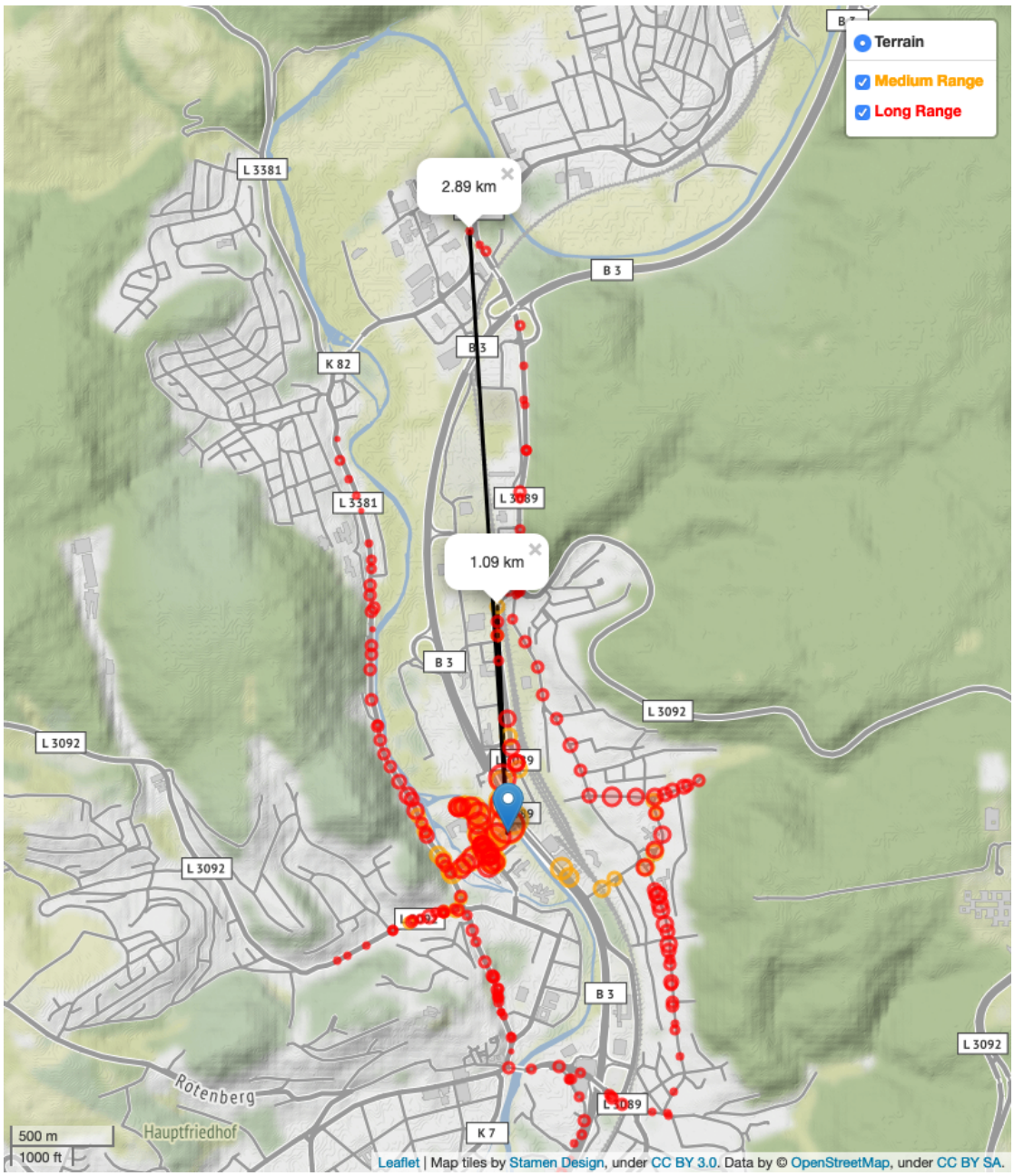


Figure 5.45: Successful LoRa transmissions in the city area.

In general, this shows that LoRa is a viable option to enable device-to-device communication in crisis scenarios, where infrastructure is destroyed or temporarily not available. The different profiles of LoRa can be used to limit communication to a certain area and therefore allow higher data rates, or cover a larger area and therefore reach out to more people.

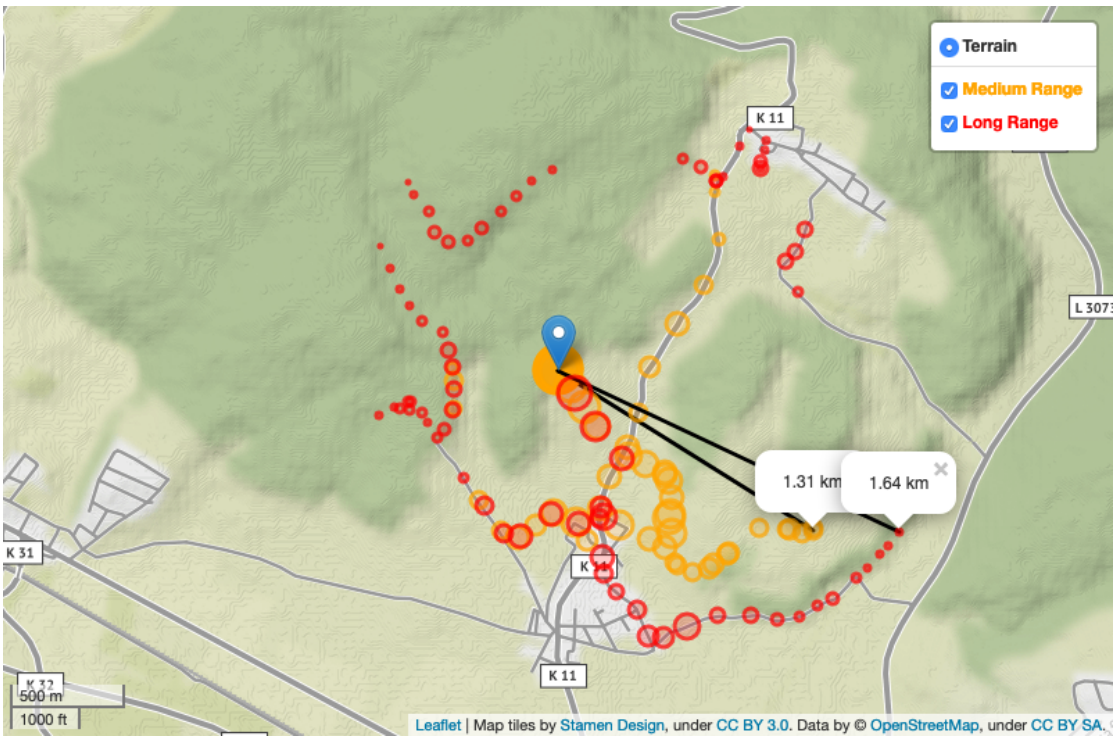


Figure 5.46: Geo-positions of successful LoRa transmissions in a rural area.

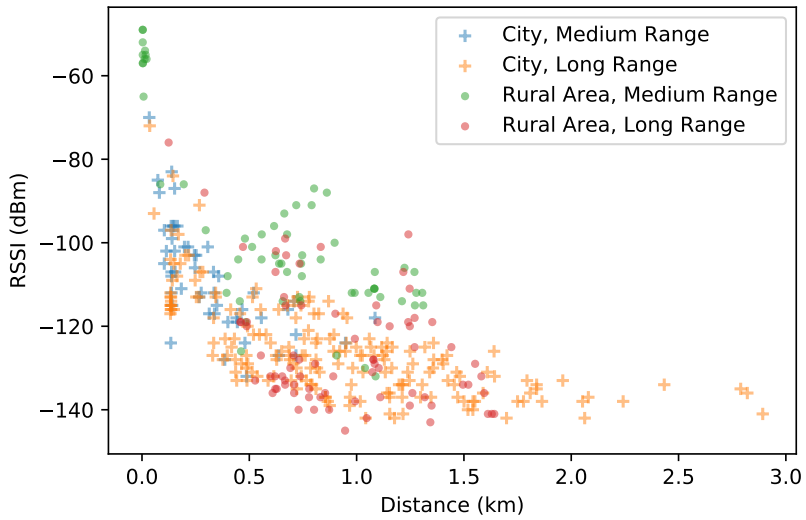


Figure 5.47: Received Signal Strength Indicator in relation to transmission distance in the proposed device-to-device scenario.

Interfacing Emergency Networks

When transmitting data over a disruption-tolerant network, an overhead is generated. This is caused by the additional meta-data that a DTN bundle carries, e.g., the sender, receiver, or other blocks of information. In addition, there is now a second overhead for the fragmentation header of the BBC. Due to the small size of a LoRa packet, it is advisable to examine the total size of a transmission and the number of fragments. The benefits or costs of the `xz` compression should also be considered.

For our evaluation, we created two types of payload data: randomly distributed data and the *lorem ipsum* placeholder text. The respective payloads were generated in the sizes of the power of two, from 2^1 to 2^{11}. For this purpose, the 445 byte long lorem ipsum text was shortened or repeated accordingly. This payload was wrapped into a DTN packet, sent from `dtn://source/` to `dtn://destination/` with an additional age block to set the lifetime to one hour. The LoRa maximum payload can be up to 251 bytes in size, as instructed by *rf95modem* in our test configuration.

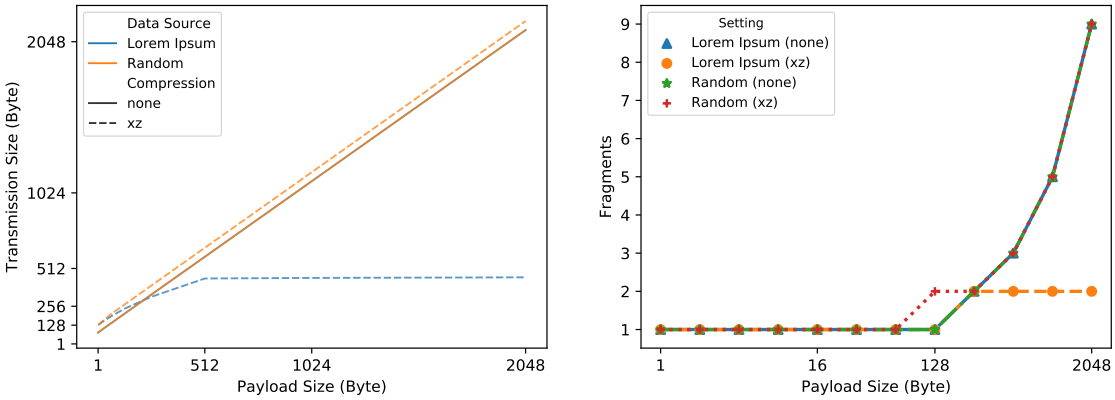


Figure 5.48: Total transmission size and amount of fragments for different payloads.

The overhead of a DTN bundle is 77 bytes without compression. In Figure 5.48, the final transmission size and the number of required fragments are shown for the two characteristics of the payload data and its size. It is noticeable that for a random payload the transmission size is slightly larger. However, the number of fragments is almost always the same. Furthermore, user data is usually not randomly distributed. This is where the advantage of the compression comes into effect, as it becomes evident especially in the low number of fragments with compressed payloads.

We also carried out a small field test. For this purpose, three DTN nodes were installed, each equipped with a *rf95modem* for 868 MHz in the Short Range profile: 500 kHz, Cr: 4/5, SF7. The nodes were positioned so that only one node had direct radio contact with the other two. Every time a packet is forwarded in a DTN, some meta-data is updated, e.g., the Previous Node to specify the last relaying node. To verify the packet forwarding, we inspected the Previous Node from the received packet. If this value does not match the packet's sender, it was successfully forwarded. To perform this evaluation, we prepared three nodes, n_0, n_1, and n_2. n_1 was positioned in the midpoint, further n_0 and n_2 were not supposed to have direct contact. Outgoing from n_0, packets were sent addressed to n_2. These should be transferred

from n_0 to n_1 first and forwarded from n_1 to n_2 afterwards. We then sent DTN packets with a small payload so that they fit into a single LoRa packet. As a result, we observed situations where the Previous Node was adjusted accordingly. In such a case, the round trip time took 1.7 seconds from initiating the transmission to receiving the acknowledgement of reception.

Energy Considerations

While the energy consumption of smartphones is a well studied field and battery lifetimes of these devices are up to some days, the companion devices studied in this section are not evaluated that well. Thus, we measured multiple devices targeted by the proposed firmware in terms of energy usage in different energy states, namely receiving, sending, and deep sleep. From these measurements, the required battery capacities can be inferred.

Board	Receiving	Sending	Deep Sleep	Additional Features	Price
TTGO T-Fox 27 dB	392 mW	1,771 mW	61 mW	WiFi, BLE, OLED, RTC	20 €
TTGO T-Fox 20 dB	400 mW	902 mW	61 mW	WiFi, BLE, OLED, RTC	20 €
TTGO LORA ESP32	404 mW	782 mW	68 mW	WiFi, BLE	15 €
TTGO LORA32 V2.0	393 mW	689 mW	57 mW	WiFi, BLE, OLED, SD	20 €
TTGO LORA32 V2.1_1.6	387 mW	785 mW	57 mW	WiFi, BLE, OLED, SD	20 €
Heltec Wireless Stick	391 mW	923 mW	76 mW	WiFi, BLE, OLED	12 €
Adafruit Feather 32u4 LoRa	72 mW	648 mW	49 mW	-	35 €
Adafruit Feather M0 LoRa	95 mW	697 mW	<1 mW	-	35 €
TTGO T-Beam v0.7	723 mW	1,125 mW	393 mW	WiFi, BLE, GPS	20 €

Table 5.11: Energy consumption in receiving, sending, and deep sleep modes of *rf95modem* compatible boards

The energy consumption was measured using an ODROID Smart Power Meter[57] connected to the microUSB connector of the board and supplied 5 V.

In Table 5.11, the average energy consumption of the listed boards is presented. Since the boards need to be online to receive messages from other boards, the receiving mode has the highest impact on energy consumption.

The power consumption of the measured boards when receiving data shows a broad variance, e.g., from about 72 mW for the Adafruit Feather 32u4 LoRa board up to 723 mW for the TTGO T-Beam v0.7 board. While sending data, the required power differences become more balanced. When deployed in sensor networks, the deep sleep power consumption becomes important. Four of the tested boards require 49 to 76 mW in this mode, while one board requires below 1 mW. The values for deep sleep are likely caused by powering the boards through the microUSB connection, which requires a transformation to the voltage required by the microprocessors. Also, most boards contain a serial to USB converter, which cannot be turned off when powered via USB.

[57] `https://www.hardkernel.com/shop/smart-power/`

To put these numbers into perspective, we assume a powerbank with a capacity of up to 20,000 mAh. Such powerbanks are widespread and used by smartphone users to recharge their phones. This capacity at 3.3 volts relates to 66 Wh, and thus can power the TTGO and Heltec hardware for more than 160 hours. The maximum receiving time can be achieved using the Feather 32u4 LoRa board with more than 900 hours of receiving time.

Scalability

When speaking of LoRa, the question regarding usability with respect to large networks and radio interference arises. As mentioned earlier, LoRa, or more precisely any protocol in the same frequency band, must follow a strict duty cycle of 1% with respect to time. This is mainly to allow for fair use of the radio spectrum and to reduce collisions of packets leading to data loss. In an emergency scenario with users sending messages in an uncontrolled and unrestricted manner, however, it is hardly possible to enforce any such limitation. Thus, in this section we investigate the limitations of LoRa with respect to three aspects: (a) how many active users can be in the network before rendering it unusable due to too many collisions, (b) how many people can be reached in which distance (i.e., how far do LoRa packets travel), and (c) what can be done to circumvent saturated networks with respect to practical applicability.

Experimental Setup To perform large-scale tests with a high number of devices sending data using LoRa, we rely on the NS-3 network simulator [RH10]. NS-3 allows us to simulate a high number of users using different physical layer implementations as well as a variety of path-loss- and propagation models. However, currently, NS-3 does neither support LoRa as the physical layer nor the LoRaWAN data link layer. Thus, we use the LoRaWAN plugin for NS-3 presented by Magrin et al. [MCV17]. By omitting the data link layer implementation and sending data directly to the physical layer, the used LoRaWAN plugin can also simulate the LoRa physical layer without the LoRaWAN data link layer, which emulates the usage of our proposed application.

Due to the duty cycle requirements of LoRa, one main goal of this test is to explore how our system performs under different amounts of network traffic. Furthermore, it is more likely that such a LoRa communication application as proposed in this paper is reaching its limits in urban environments than in rural areas due to the different population densities. Thus, we modeled a city including suburban areas of 10 km x 10 km. We assume a population of 100,000 inhabitants, whereas their distribution follows roughly a normal distribution on both sides of the square, where the mean is set to the center (5,000) and the standard deviation to 1,000. This results in a population distribution that is densest at the center of our hypothetical city and decreases with higher distances. Figure 5.49 visualizes this distribution. Each cell in the grid represents a 100 by 100 m square, where an empty cell is blue and a more crowded cell gets brighter. We used this distribution since it roughly resembles the distribution of a city: many people live and spend their time in the city center, while the outer areas of a city, i.e., the suburbs, are populated less densely. Regarding the number of users, we assume that realistically at most 1 % of the population would use such an application. Thus, we simulated scenarios of 100 (0.1 % of the population), 500 (0.5 % of the population), and 1,000 users (1 % of the population). Finally, we modeled three different user behaviors: users sending only

Parameter	Values
Simulation time	1 hour
Area	10 km x 10 km
Seed	35039
Repetitions per configuration	5
Base frequency	868.0 MHz
Users	100, 500, 1000
Messages per user	3, 10, 50
LoRa configurations (SF, BW, Payload)	1. SF7, 250 kHz, 222 bytes 2. SF7, 125 kHz, 222 bytes 3. SF7, 125 kHz, 51 bytes 4. SF9, 125 kHz, 51 bytes 5. SF12, 125 kHz, 51 bytes

Table 5.12: Experimental configurations

a few messages (3), e.g., because they are currently helping others or because they are busy doing other things during an emergency. On the other end, we modeled users sending many messages (50), e.g., because they are actively searching for people. Finally, an average user was modeled to send 10 messages. During a simulation, each user sends the specified number of messages during the simulation period of one hour, where the time a user sends its messages is uniformly distributed across the entire simulation time.

The next parameter set refers to the LoRa parameters. For the base frequency, we used 868.0 MHz since it is predominantly and almost exclusively used in Europe. Furthermore, to test the capabilities of different LoRa settings and their effect on both the maximum transmission distance as well as interferences under high loads, we used five different configurations: 1. SF7 with a bandwidth of 250 kHz using a payload size of 222 bytes, 2. SF7, 125 kHz, and 222 bytes payload, 3. SF7, 125 kHz, and 51 bytes payload, 4. SF9, 125 kHz, and 51 bytes payload, 5. SF12, 125 kHz, and 51 bytes payload. These payload sizes were chosen as they are both the maximum payload size of a LoRa packet for the given configuration (except configuration 4.) and, at the same time, also provide a good reference size for typical short messages. Table 5.12 summarizes these parameters.

Furthermore, each experiment was repeated 5 times, to cope with side effects due to unfortunate randomness, e.g., during user positioning. Finally, as the experiments require randomness for distributing the users within the simulation area and selecting sending times within the simulation time, we used a starting seed of 35039 and incremented this number for every iteration of the 5 simulation repetitions resulting in 805 overall simulation runs.

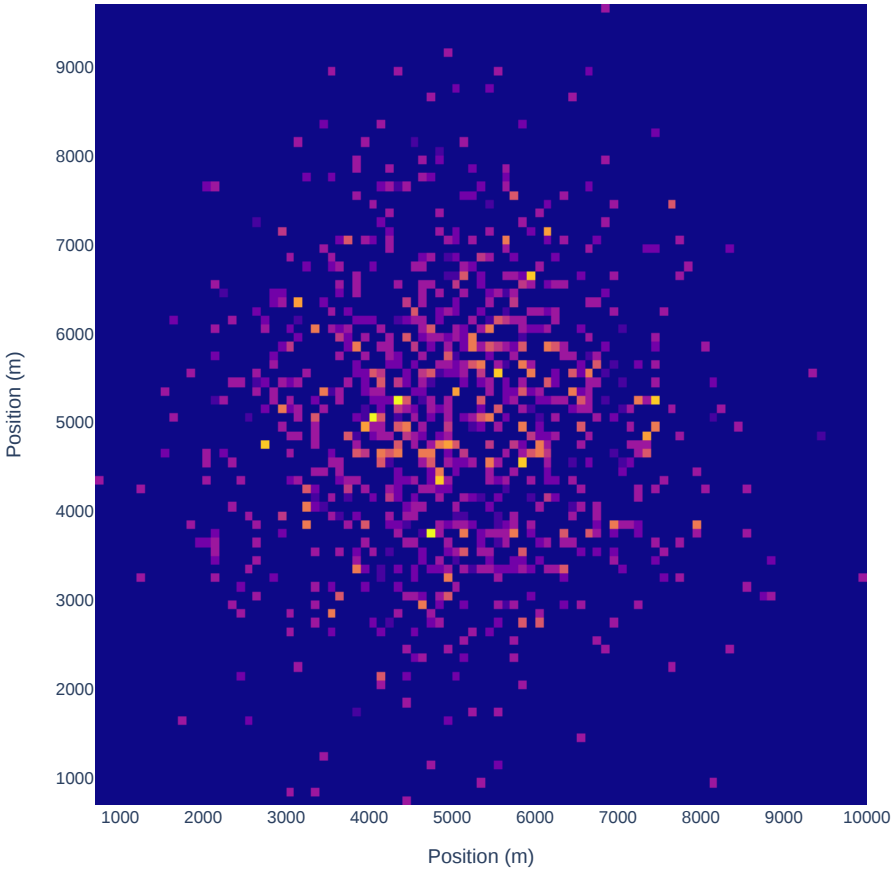


Figure 5.49: User Distribution.

Applicability and Limitations Since LoRa is intended to be used as a low-bandwidth technology, one of the primary questions one needs to ask when considering feasibility is at what point will traffic saturate the network.

In Figure 5.50, each plot represents one distinct simulation parameter set, as described in the previous section. Each row containing three figures shows results for different messages per user, each column represents a different number of users. Within each figure, each bar on the x-axis shows a different LoRa configuration with respect to SF, bandwidth and payload and the y-axis shows the percentage of attempted transmissions which resulted in one of five states. Note that due to the broadcast nature of LoRa, a single transmission will lead to n-1 reception events (where n is the number of users) with potentially different results:

- Success represents successful transmissions, i.e., a user received the packet and was able to successfully decode it.
- Failure (Signal Strength) represents users being unable to receive a packet because the signal attenuation due to path loss was too high.
- Failure (Interference) is an unsuccessful reception due to multiple, simultaneous transmissions interfering with each other.

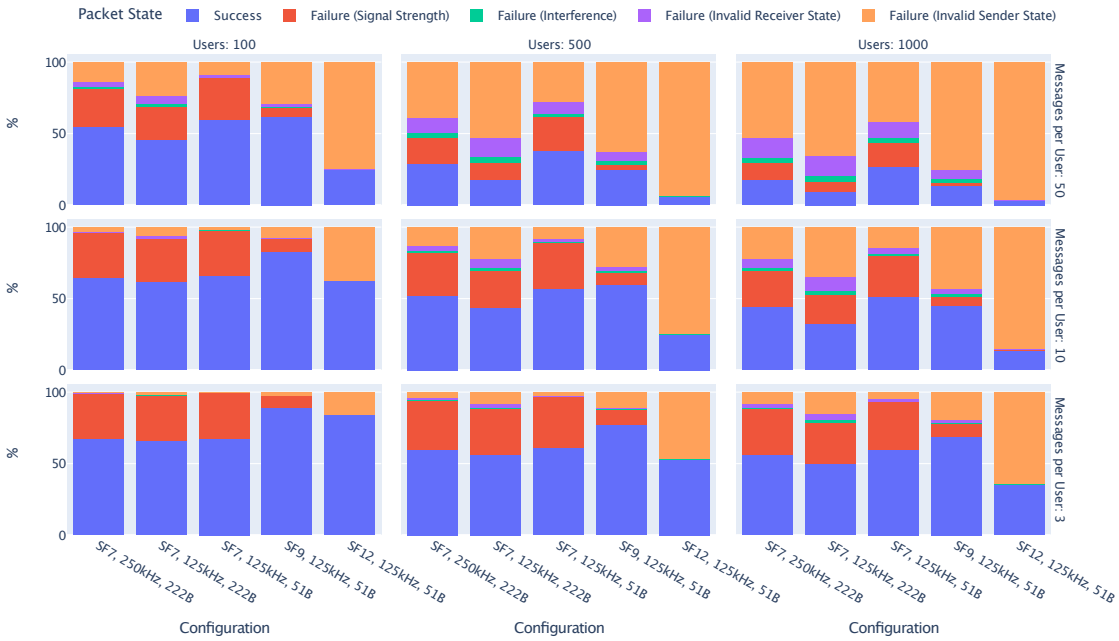


Figure 5.50: Transmission Results.

- Failure (Invalid Receiver State) means that the receiving LoRa module was in a state in which it was unable to receive the packet. This occurs since LoRa modules cannot simultaneously transmit and receive data.
- Failure (Invalid Sender State) is a packet reception that did not occur because the packet was not sent at all. This can be due to either one of two reasons. One possible reason is the sender being in receive mode when the packet was meant to be sent. Since LoRa modules cannot send data while they are receiving, this results in a failure. The other possible reason for this failure mode is that a LoRa module can only send a single packet at once, thus if one tries to send a second packet while the first is still being transmitted, the sending fails.

It can be observed that with increasing load, be it due to a higher number of users, or more packets sent per user, or both, the probability that a packet will be received successfully decreases. While an increase in messages seems to impact delivery somewhat more strongly than an increase in users, the difference seems comparatively small. However, as either, or both, load factors increase, the delivery probability quickly drops to or below 50 %.

Interference between senders plays virtually no part in the measured decrease of the delivery ratio. Rather, the majority of unsuccessful transmissions are due to invalid hardware states, with an invalid sender state quickly becoming the vast majority of failure conditions, followed by an invalid receiver state. Failures due to signal attenuation do not change in any meaningful way in between load scenarios, which is to be expected since greater or smaller loads do not change physical constraints that are responsible for these failures. Only the most congested scenarios result in a significant decrease of this type of failure, but this is most likely simply due to the overwhelming effect of the invalid state failures that overpower all other conditions.

The only parameter that has a major effect on the delivery range is the spreading factor, with SF9 having already greatly decreased signal-strength failures, and SF12 effectively eliminating them altogether. However, the tradeoff of higher spreading factors is obvious, since they are the most susceptible to state failures as the load increases. Comparing the three rightmost columns, which are identical except for different SFs, it becomes obvious that any spreading factor greater than seven is infeasible for our use case with respect to the ratio of successfully received packets compared to failed transmissions. Higher spreading factors increase the time it takes to send the same amount of data. Transmitting a 222 bytes large packet using SF7 and 125 kHz bandwidth takes roughly 370 ms whereas sending 51 bytes using SF12 requires about 2800 ms airtime, i.e., 7.6 times more. This also increases the likelihood of the LoRa module being occupied in the sending state where it can neither receive packets nor accept new packets for transmission. The fact that sending takes longer in SF12 explains this observation.

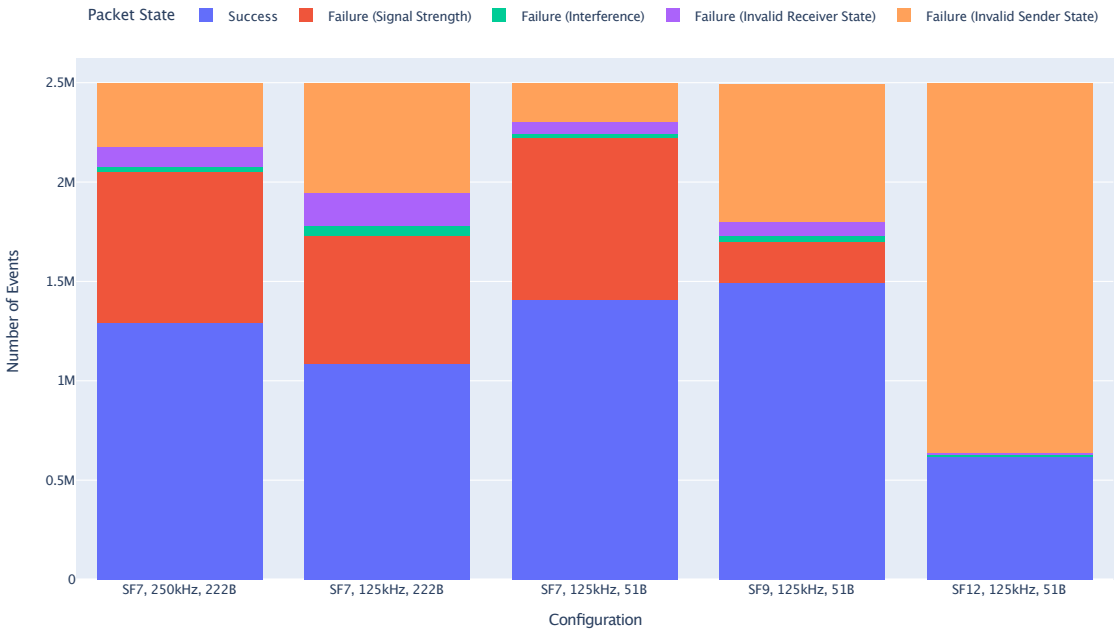
Figure 5.51 is generated from the same data as Figure 5.50 but shows the total number of events, rather than the percentage-based normalized values in Figure 5.50. The differences in load that are separating the simulation scenarios can be best understood when having this view on the data.

To summarize, it can be seen that the probability of successfully delivering a packet is highly susceptible to network congestion. To cope with this challenge, we need to find mitigations that allow us to prevent saturating the LoRa band, one of which we are going to present in the following section.

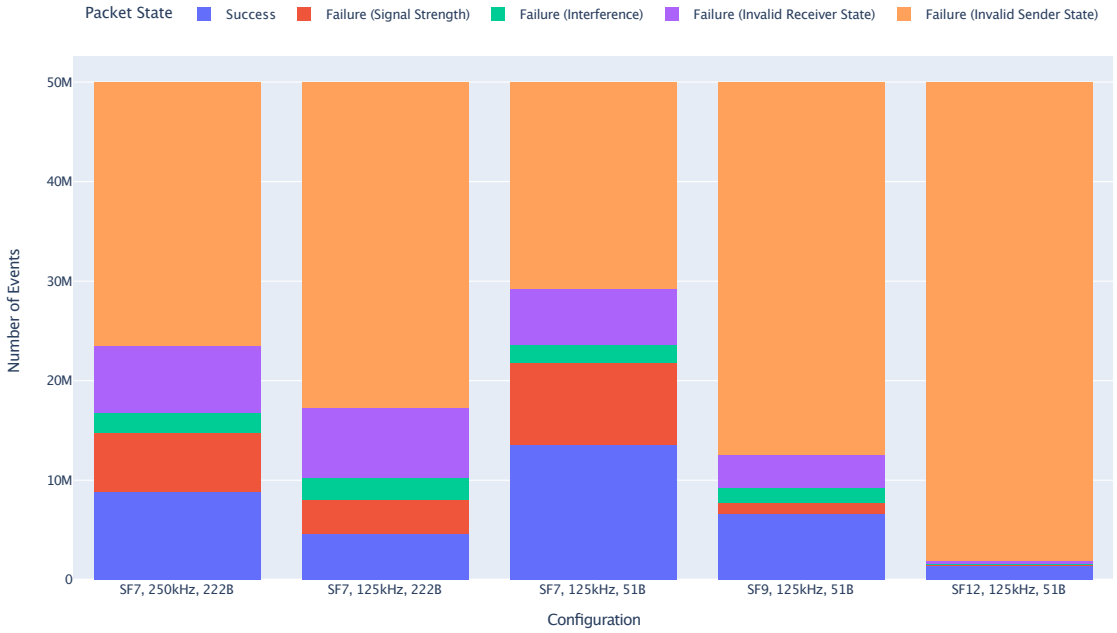
While congestion may be the principal issue in the way of real-world feasibility, transmission distance is another. Since LoRa messages are single-hop broadcasts, if two users are too far apart for direct transmission, they can effectively not communicate. Therefore, we have to answer the question of how far LoRa packets get depending on the experimental configuration.

Figure 5.52 shows the reception events in their spatial distribution, where the meaning of the colors of the packet states is the same as in the above figures. Note, that in Figure 5.50 (Transmission Ranges) the failure (Invalid Sender State) is not shown because packets that could not be sent do not have any location information and thus no distance associated. Furthermore, the x-axis denotes the LoRa configuration, where every group of boxes is associated with one LoRa configuration, and the y-axis denotes the distance in meters that a packet traveled between sender and receiver.

One insight of the evaluation is that the general results of the distance evaluation do not depend on the load of the network, i.e., they are largely independent of how many users are sending in the network and how many packets each user sends. Thus, Figure 5.52 only shows distances of packets for a single number of users (500) and a single configuration for the messages per user (10). SF, bandwidth, and payload are set as discussed previously. As can be seen in Figure 5.52, the configured bandwidth and packet payload do not affect the distance of a transmitted packet. The first three groups show SF7 but with different bandwidths and payloads. The distance of successful transmissions, however, does not change. With a lower bandwidth of 125 kHz interferences occur after a slightly shorter distance, but only visible in outliers, while the quartiles and medians do not differ significantly. The difference between the payload with SF7 and 125 kHz bandwidth with respect to interferences can also be seen in the outliers of the green boxes of groups 2 & 3. Here we can see that a smaller payload results



a) 10 messages per user, 500 users.



b) 50 messages per user, 1,000 users.

Figure 5.51: Transmission Results (Absolute).

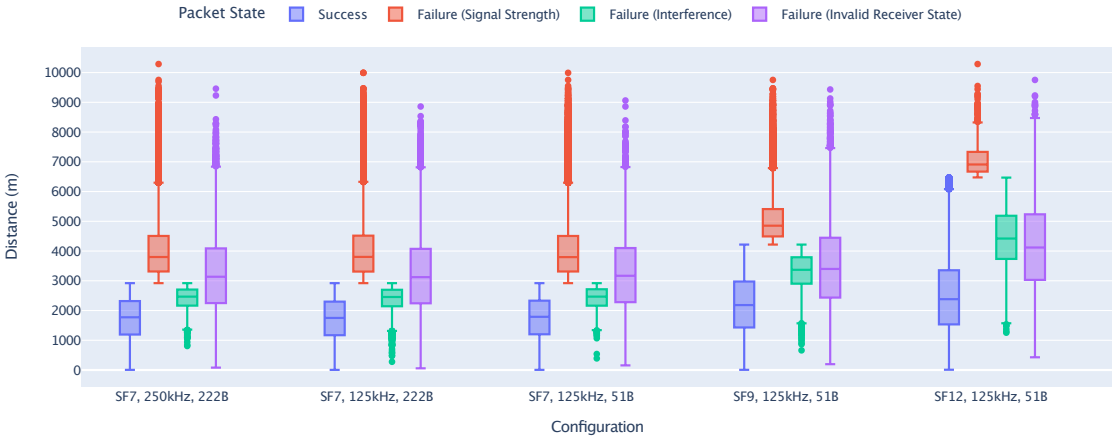


Figure 5.52: Transmission ranges for 500 users and 10 messages per user.

in less interference, which is validated by the above evaluation of the transmission results. The most influential factor with respect to transmission distances is the SF. Higher SFs result in farther successful transmissions and fewer failures due to low signal strength. However, this increase in transmission range also leads to a bigger area where interference can occur, as evident in the last group of boxes representing SF12. This is explainable by the fact that higher SFs result in an increased airtime. Thus, for SF12 it is more likely that interferences occur, which is also reflected in the increased distance of interferences. The same argument also applies for an increased range of failures due to invalid receiver states. The longer the transmission takes, the higher is the probability that a user is currently in the sending state and not able to receive an incoming packet across all distances.

In summary, these results show that LoRa is able to cover a large area of a city. Users are able to reach other users within a radius of up to 2.9 km for SF7, 4.2 km for SF9 and 6.4 km for SF12. Due to interferences in the ranges around 2.4 km (SF7), 3.3 km (SF9) and 4.4 km (SF12) on the average, the usable radius is about 1.7 km, 2.2 km and 2.5 km, respectively. However, it must be noted that the average transmission range is a result of our user distribution. With a different geographic distribution of users, the mean transmission range also changes due to interferences in different distances, but not the maximum. These results show that LoRa and especially our approach is suitable to provide emergency communications with respect to the communication range. With this transmission range, people in affected areas can communicate, coordinate themselves, and ask for help with a high chance to reach first responders that might not be in proximity, but are still able to receive messages due to the high LoRa transmission range.

Building LoRa Communities As a result of the issues discussed in the previous section, a single LoRa channel is of limited use to a city public, such as for the distribution of public information or emergency messages. Luckily, the different international frequency bands available for LoRa provide us with a way to implement multiple, non-interfering channels, which can be used for better practical usability in a scenario as ours. In addition to these separately usable frequencies, chirp spread spectrum modulation has the advantage that the spreading factors are orthogonal, which means that messages sent with one spreading factor

do not interfere with the transmission of messages from another spreading factor. Following the LoRa Alliance's definition of 8 channels in the 868 MHz band and the common spreading factors 7-12, a total of 48 independent channels are available. These channels can be used by different communities and institutions, whereby the channel distribution is either agreed upon in advance or negotiated among the users at a central coordination channel.

Parameter	**Values**
Users	100
Messages per user	1, 10, 20, …, 200
LoRa configurations (SF, BW, Payload)	SF7, 125 kHz, 222 bytes (cf. 2.) SF9, 125 kHz, 115 bytes SF12, 125 kHz, 51 bytes (cf. 5)

Table 5.13: Experimental configurations for additional edge-case tests

To get an impression of the usability of a single channel, we performed additional simulations in which the channel was used by 100 users with different message rates (1 - 200 messages per user and hour). For this experiment series, we used spreading factors 7, 9, and 12, and their respective maximum message lengths. A summary of the updated values used for these tests can be found in Table 5.13.

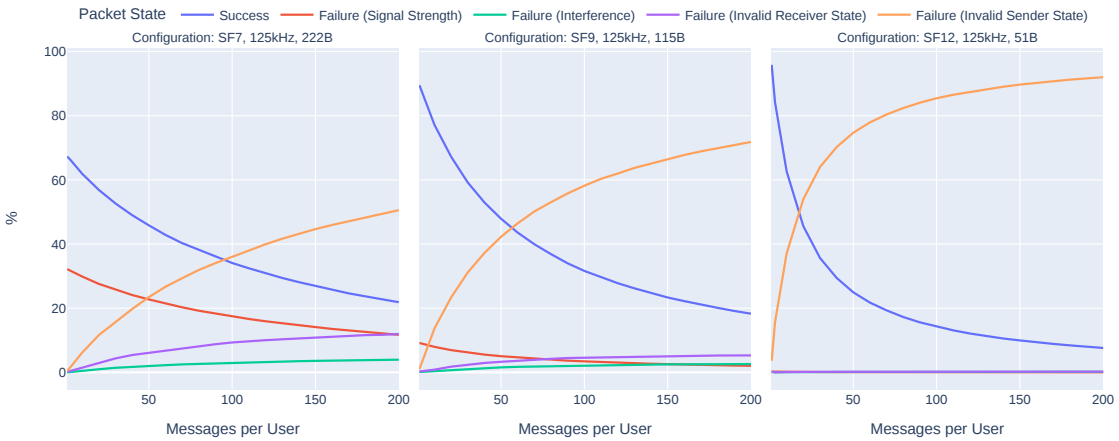


Figure 5.53: Message receiving performance for different spreading factors and variable messages per user for a community of 100 users.

Figure 5.53 shows the experimental results of the proposed experiment in a community of 100 people. As already indicated in the previous experiments, the rate of successfully delivered messages is limited by the range, especially in smaller spreading factors. For SF7, 27.6 % - 33.2 % of the messages are lost due to low signal strength, while SF9 incurs 6.3 % - 9.9 % loss. When using spreading factor 12, the transmission time of the packets is so high that a successful transmission is no longer possible even with a low number of messages per user. The capacity limit of the channel can be derived by determining the intersection of the successful deliveries and the transmission prevented by simultaneous reception, i.e., Invalid Sender State.

Following this scheme for 100 users, a SF7 channel has a capacity of 90 222-bytes messages, a SF9 a capacity of around 60 115-bytes messages and a SF12 channel is limited to around 20 51-bytes messages. These metrics, alongside with the range benefits and drawbacks of individual spreading factors can help communities to decide about a configuration to establish useful communication.

5.6.6 Summary

In this section, we presented an approach to facilitate long range device-to-device communication between smartphones in crisis situations. Our approach relies on inexpensive microcontrollers with integrated LoRa hardware that we enabled to receive and forward messages via Bluetooth, WiFi or a serial connection. We developed a dedicated firmware, called *rf95modem*, to provide this functionality not only in crisis situations, but also in several other applications, such as providing a communication fallback during outdoor activities, geolocation-based games or broadcasting of local information. To illustrate the practical relevance of our approach, we implemented a novel device-to-device LoRa chat application for iOS, Android, and laptop/desktop computers. Furthermore, we integrated LoRa using *rf95modem* into the disruption-tolerant networking software DTN7. ur experimental evaluation based on real world device-to-device LoRa transmissions in urban and rural areas, as well as scalability tests based on simulations of LoRa device-to-device usage with up to 1.000 active users showed that our approach is technically feasible and enables low-cost, low-energy, and infrastructure-less communication.

The approach presented here for integrating LoRa into existing devices is not to be understood directly as a smart system in the sense of this thesis, but rather as a support for the further development of smart systems. The integration into DTN7 and a scenario-specific implementation or optimization based on it using LoRa technology could then be considered a smart solution in the sense of this thesis.

6

Smart Transitional Wireless Networking

In this chapter, novel approaches in the field of smart transitional wireless networking are presented. In traditional networking applications, a certain protocol combination is selected during development and used under quite different circumstances. In contrast to parameter adaptation, which is a common practice, the replacement of a mechanism by another mechanism with better performance and quality is a new paradigm.

A fundamental requirement for transitional networks is the decision basis on which transitions are performed, in particular the classification of network traffic flows. A novel data-driven network traffic flow classification approach based on statistical properties of individual network flows is described in Section 6.1.

Service announcements, such as in adaptive peer-to-peer networks discussed in the previous section, often rely on fixed announcement intervals. Section 6.2 presents multiple approaches to realize dynamic announcement intervals, to reach the goal of fast reception from at least one node, while trying to keep overall communication overheads low.

In Section 6.3, a data-driven approach to perform Wi-Fi/cellular transitions is presented. Data provided by multiple smartphone sensors, e.g., Wi-Fi RSSI, acceleration, compass, step counter, air pressure, are used to predict Wi-Fi connection losses and transition to cellular connections.

The achievable quality and information analysis cost of the approaches presented in this chapter are shown in Figure 6.1. The information analysis cost of the contributions presented in the area of smart transitional wireless networks is particularly associated with computations. The smart use of already existing data with the help of modern data-driven and machine learning based algorithms allows to increase the quality of a system or algorithm. In the three contributions, different quality metrics are considered. In the case of dynamic announcement intervals, the metric is the delay achieved for the discovery of a group of network peers, i.e., the QoS. In unsupervised traffic flow classification, the QoR is improved, specifically precision and recall of the compared algorithms. In seamless vertical handovers, the QoE perceived by users is derived and optimized. In the case of video streams, these are stalling events and video quality adaptation, which are decisive for the perceived quality, according to the recognized models.

Contributions to `pkt2flow` used in Section 6.1[1], as well as collected data, trained models and a demo application of the data-driven handover mechanism[2] are released under open-source licenses.

[1] `https://github.com/jonashoechst/pkt2flow`
[2] `https://umr-ds.github.io/seamcon/`

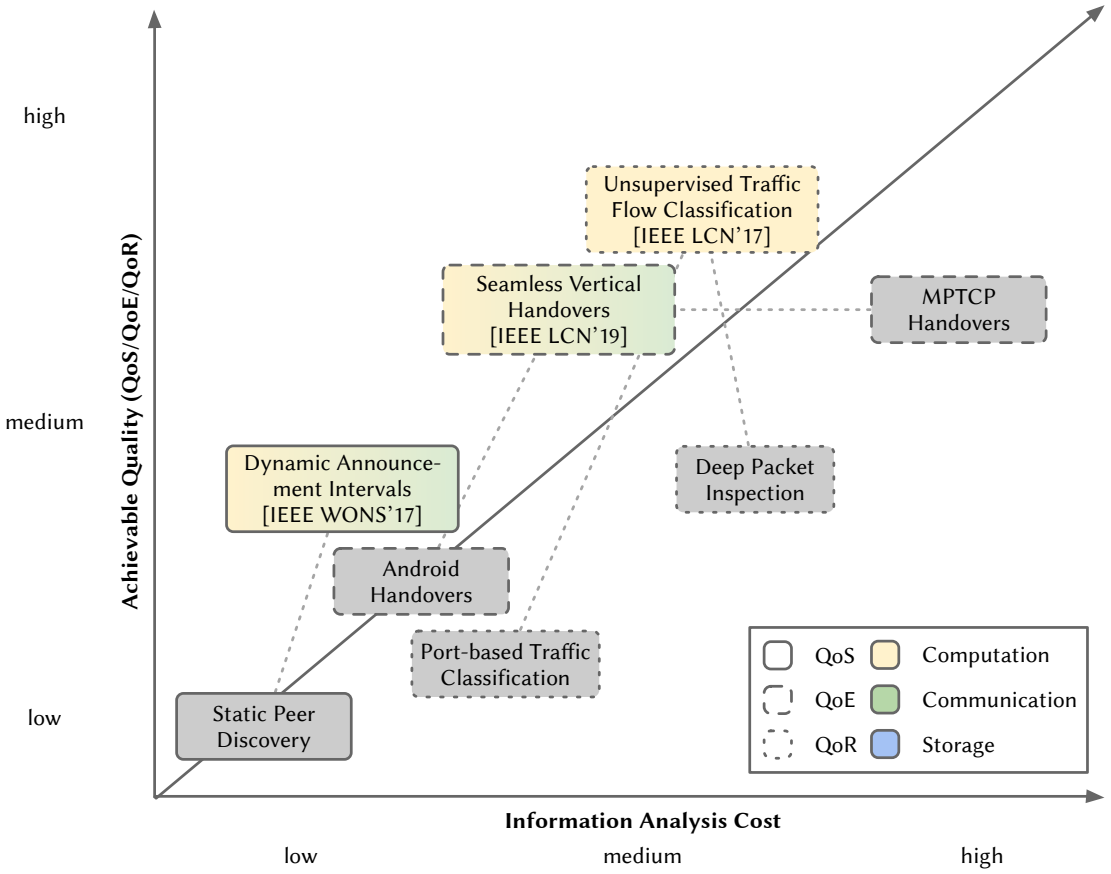


Figure 6.1: Information analysis cost and achievable quality of smart transitional wireless networking approaches

6.1 Unsupervised Traffic Flow Classification Using a Neural Autoencoder

6.1.1 Introduction

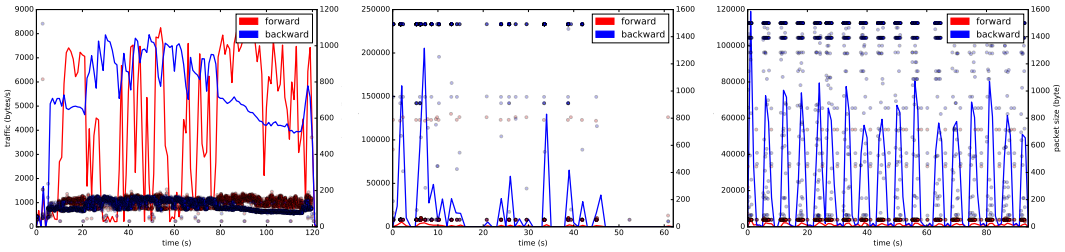
The growing popularity of smartphone and tablet usage poses challenging demands on mobile communication. Apart from browsing the web, real-time and high-bandwidth services, such as Voice-over-IP (VoIP) or video live streaming, are increasingly used in today's mobile applications. With the paradigm shift towards Software-Defined Networking (SDN) in the network core and Software-Defined Wireless Networking (SDWN) in the network edge [Ber+14], dynamic flow configurations can heavily profit from network traffic classification for priorization and resource management.

Currently, Internet traffic classification is used to improve Quality-of-Service (QoS) or Quality-of-Experience (QoE) [NA08; Li+13]. Traffic classification methods can be grouped into port-based, payload-based and statistical methods [DPC12]. Since many applications do not rely on

fixed port numbers and ports can be easily redirected or obfuscated, port-based methods are inadequate for characterizing the properties of network traffic.

Payload-based methods use certain fields of application layer protocols to classify traffic. In particular, Deep Packet Inspection (DPI) uses fixed protocol signatures of the packet payload [Li+13; Qin+15]. However, payload-based methods fail whenever connections are encrypted. Furthermore, they can be easily circumvented, and changes of the application protocols may also lead to false classifications.

Statistical approaches [MZ05; Erm+07; Zha+15b; Zha+13] typically use packet inter-arrival times, packet sizes and their statistical properties (e.g., average, maximum, minimum). They often rely on machine learning algorithms for performing traffic classification, in particular supervised learning algorithms that require labeled data for training to build a suitable classification model. The labels are often obtained from validated application ports or other labeling mechanisms (such as DPI), resulting in classification mechanisms that replicate the labeling methods, rather than revealing new structures independent of prior knowledge. In many cases, web browsing is equated with HTTP(S) traffic, while Skype traffic is assigned to video conferences. This is misleading in various ways, since nowadays HTTP(S) is the basis for many applications other than browsing the web.



a) Audio call b) Website interaction c) Buffered videostream

Figure 6.2: Traffic utilization and packet sizes of example flows.

In this section, a novel approach to unsupervised traffic flow classification using statistical properties of flows and clustering based on a neural autoencoder, is presented. In contrast to previous work, the neural autoencoder is used to automatically cluster traffic flows, e.g., into downloads, uploads, or voice calls, independent of the particular network protocols, such as FTP or HTTP(S), used for performing these tasks. A novel time interval based feature vector construction is proposed. By separating flows into exponentially growing time periods and computing their statistical properties individually, a weighted bandwidth graph is fed into the neural autoencoder. A semi-automatic cluster labeling method facilitates traffic flow classification independent of known traffic classes. An experimental evaluation on real data captured from about 25 mobile devices performing daily work over a period of 4 months is presented. The obtained results show that 7 different classes of mobile traffic flows are detected sufficiently fast with an average precision of 80% and an average recall of 75%. When only specific classes for quality-of-service optimizations are considered, F1 scores of over 90% are achieved. Thus, our main contributions are:

- We present a novel unsupervised machine learning approach based on a neural autoencoder for network traffic flow classification.
- We propose a novel feature extraction method relying on statistical values in exponentially growing time slots of flows.
- We suggest a novel protocol-independent cluster labeling and classification approach suitable for QoS/QoE optimization at the network edge.
- We describe an efficient implementation of the neural autoencoder together with an experimental evaluation to demonstrate the feasibility of our approach.

Parts of this section have been published in Jonas Höchst, Lars Baumgärtner, Matthias Hollick, and Bernd Freisleben. "Unsupervised Traffic Flow Classification Using a Neural Autoencoder." in: *42nd Annual IEEE Conference on Local Computer Networks (LCN 2017)*. Singapore, Oct. 2017. DOI: 10.1109/LCN.2017.57.

6.1.2 Related Work

The survey of Nguyen and Armitage [NA08] compares the results of various machine learning approaches applied to network traffic classification. Traffic classification based on machine learning is still an open field [NAS15].

Classification approaches based on multiple identification methods reaching from packet headers to full flow examination and even host history inclusion have been presented [MP05; Erm+07] to classify traffic into 10 classes, such as INTERACTIVE, MAIL, WWW or MULTIMEDIA. Moore et al. [MZ05] use Bayesian analysis techniques with 248 per-flow features to reach a basic classification accuracy of 65%. By improving the basic methods, 95% classification accuracy is reached.

Kim et al. [Kim+08] criticize the high variability in data sources and classification targets. The authors propose to use *overall accuracy*, *precision*, *recall* and *F-Measure* as performance metrics, as well as 4 different publicly available datasets created between 2004 and 2006. They also compare different machine learning approaches and reach 94.2%-97.8% accuracy using a Support Vector Machine (SVM) [VC74].

Semi-supervised methods have been proposed [Erm+07] to find clusters of traffic flows using the K-means algorithm [Llo82]. The clusters are labeled afterwards using a small set of labeled flows. The authors also propose a real-time classification method where multiple layers offer classifications based on *packet milestones*. Vladutu et al. [VCD16] present a semi-supervised framework for flow classification using generated traffic and thus the resulting semantic categories.

Zhang et al. [Zha+13] present an unsupervised clustering algorithm based on statistical properties of flows as well as payload-based clustering. The authors use 13 different classes as their ground truth, made up of different protocols, such as HTTP and SSH. The flows are clustered using several configurations of the K-means algorithm. The generated clusters are labeled by the application flows dominating the specific cluster. Using this method, the authors reach an accuracy of over 90%.

name	description	unit	default
packets	# packets	-	0
bytes	# bytes	-	0
bytes_avg	avg. packet size	byte	0
bytes_std	stdev. packet size	byte	1
iat_avg	avg. packet inter-arrival	s	0
iat_std	stdev. packet inter-arrival	s	1
traffic_avg	avg. speed	byte/s	0
traffic_std	stdev. speed	byte/s	0
dscp_median	median DCSP flag	-	0

Table 6.1: Statistical flow properties

The majority of methods proposed in the literature are based on supervised learning methods. Using unsupervised clustering instead, the methods do not rely on the ground truth of the labeling mechanisms. Clearly, at some point labels have to be attached to be able to compare the mechanisms, but the actual learning is independent of pre-labeled flows that are hard to obtain in good quality and/or high numbers [NA08; Zha+13].

6.1.3 A Neural Autoencoder for Traffic Flow Classification

This section presents the design of the proposed neural autoencoder for traffic flow classification. First, the process of crafting useful feature vectors is discussed. Then, the design of the neural autoencoder for clustering is presented. Finally, we describe the mapping between the found clusters and labels relevant for our use case.

Our use case is motivated by the advent of SDN and SDWN where traffic classification is no longer useful only on central network devices, but also in the network edge and end user devices. Network properties like wireless access can be switched dynamically if the flows' properties are known. To achieve network edge traffic classification, the used algorithms need to be computationally inexpensive, since most mobile end user devices have only limited resources (e.g., WiFi routers, smartphones, 3G/4G routers).

Feature Vector Construction

We try to keep the number of statistical features low to reduce computational demands and memory usage. In addition to statistical features, such as number of packets/bytes, avg./stdev. packet size and inter-arrival time etc., the median of the *Differentiated Services Codepoint* (DSCP) field is used. This IP header field is a successor of the Type-of-Service (ToS) field and is used to indicate network demands of packets.

All features are computed in each flow direction, namely forward (client to server) and backward (server to client). For TCP flows, clients and servers are defined by the connection handshake. Since UDP is a connection-less protocol, a UDP flow is defined as a repeated exchange of packets between the same sender IP/port and recipient IP/port combination.

In Figure 6.2, three examples of network flows are presented. The red and blue lines show the forward and backward traffic. Each red and blue dot stands for a single TCP forward or backward packet, respectively. Comparing the presented examples, great differences in bandwidth consumption, inter-arrival times and packet sizes can be observed.

For Figure 6.2a and Figure 6.2c, the most important criteria for clustering are available after a short period of time. This perception can be used to improve flow clustering, in particular with respect to future online classification. To use this knowledge, the feature vector is constructed using exponentially growing time periods for statistical flow property computation. In this way, information from the beginning of flows is less reduced compared to information from the later parts. Using this method, it is also possible to constitute the duration information of flows while using a fixed length feature vector, as required by most machine learning algorithms. The feature vector is constructed using exponentially growing intervals of up to 2048 seconds.

$$(1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048)$$

There are two possible computation methods. In the cumulative method, the interval boundaries range from the beginning of the flow until the end boundary. The used values are statistical properties up to the current time. The second method is non-cumulative and therefore each interval only contains the statistical information of packets handled in the interval itself. The values of an interval are completely independent of the previous interval.

In total, each feature vector has 216 features: 2 half flows * 12 intervals * 9 statistical features.

Clustering via a Neural Autoencoder

Neural autoencoders [Son+13; Hua+14] are useful for dimension reduction and classification. The general approach of neural autoencoders is a model that is trained to reconstruct an original input vector from a smaller representation. It is trained using the squared reconstruction error as its cost function.

In Figure 6.3, our classification approach is presented. The classification approach consists of multiple steps. First, the feature vector is normalized using the standard score.

$$y_i = \frac{x_i - \mu_i}{\sigma_i}$$

While this seems to be the preferred mode, mean-only, standard deviation-only and no standardization are also evaluated in our approach, to potentially save computational overhead. The resulting vector is mainly used to train the autoencoder and afterwards only for feature

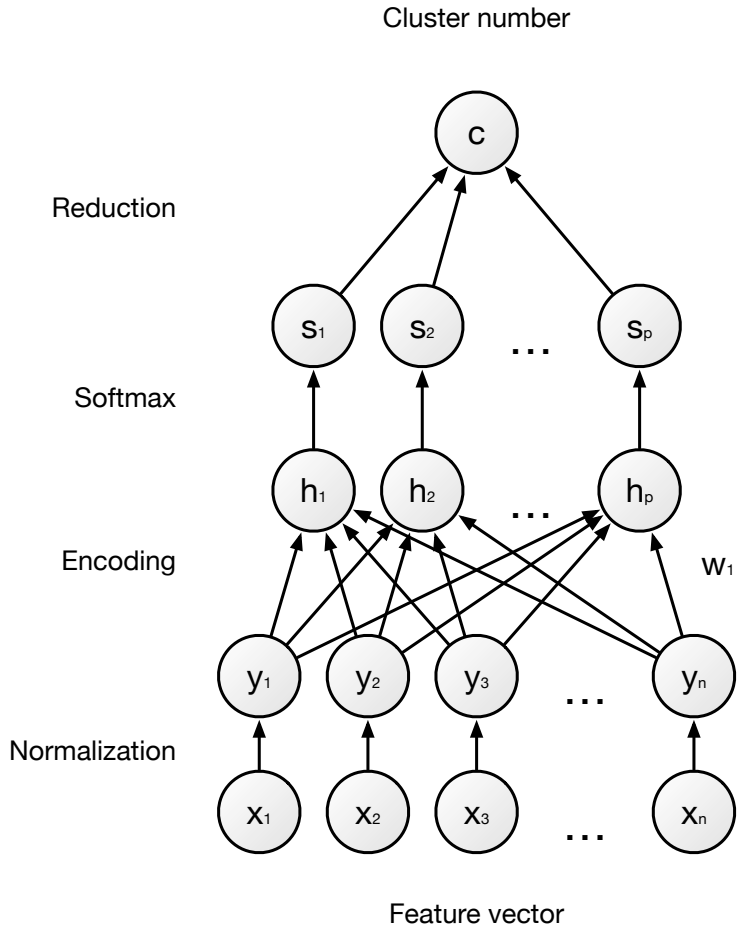


Figure 6.3: Neural autoencoder clustering.

encoding. Then, the Softmax (normalized exponential) function is applied to the encoded values.

$$s_i = \sigma(h)_i = \frac{e^{h_i}}{\sum_{p=1}^{P} e^{h_p}}$$

The actual class is then determined by selecting the index of the greatest element in the Softmax vector. This method is a rather modest classification method with the advantage of producing highly differing cluster sizes. Our autoencoder is trained using the summed squared error in combination with the Adaptive Moment Estimator (Adam)[KB15].

Cluster Labeling

The obtained clusters from the previous step need to get a semantic label. Therefore, we use a semi-automatic method comparable to Zhang et al. [Zha+13] and Vladutu et al. [VCD16]. Both labeling methods obtain a ground truth set of labeled flows. Zhang et al.[Zha+13] use

Class	Principal feature	Example mobile application
Browsing	ephemeral	Wikipedia, Spiegel, Heise
Interactive	long lasting	Online Games, Facebook, Twitter
Download	large downstream	Updates, Dropbox
Livestream	constant bitrate	Streaming, iTunes Webradio
Videostream	periodic buffering	Youtube, Vimeo, Facebook, Twitch
Call	low iat, symmetric	Skype, Apple FaceTime, Google Hangouts, WhatsApp
Upload	large upstream	YouTube, Facebook, WhatsApp

Table 6.2: Manually extracted classes

application layer protocols, while Vladutu et al. [VCD16] use generated traffic and thus the applications' names as the ground truth. Both methods have disadvantages in generalizing their classification approach, since only cherry-picked protocols are investigated.

To overcome these disadvantages, we have defined our own set of classes, based on the clusters extracted by the neural autoencoder and popular application categories. These 7 classes, their main features and examples of mobile applications, namely *Browsing*, *Interactive*, *Download*, *Livestream*, *Videostream*, *Call* and *Upload*, are presented in Table 6.2.

Figure 6.2a shows an example of a flow from the *Call* class. The primary features of a low inter arrival time and relatively low but varying packet sizes are visible in the figure. While the backward bandwidth is continuously above 5 kB/s, the forward bandwidth keeps dropping to very low values. This behavior can be explained by codec properties, such as sparse encodings of conversation breaks.

In Figure 6.2b, an example flow of browsing *Twitter* is presented. Peaks in backward bandwidth of up to 200 kB/s are visible, while the forward bandwidth stays pretty low. This graph is typical for user interactions with either social networks or even remote server management, e.g., via *SSH*. A relatively small amount of input data, e.g., scrolling, clicking, entering a command, leads to a back flow of data. The connection is held over a longer period of time, in this example for 60 seconds.

The last example is presented in Figure 6.2c. It shows the flow properties of a buffered video stream. In modern audio/video streaming protocols (e.g., Apple HLS, MPEG DASH), segments of the stream are delivered as individual files. If the local buffer runs low, another segment is downloaded. The graph of the last figure is caused by the typical bandwidth usage of this behavior.

6.1.4 Implementation

In this section, the implementation of our network traffic classification approach is presented. Our main intention is to make use of existing tools, formats and protocols where appropriate

and with portability in mind. First, an overview of how test and training data is gathered is given. Then, the separation of flows and extraction of statistics is described. Finally, classification using our neural autoencoder and automated cluster labeling are presented.

Data Capture The data used in our work was captured in an office network used by roughly 10 users including 5 frequent users with an overall count of roughly 25 devices. The data was collected in two configurations over a 4-months period: (a) by only dumping WiFi traffic of smartphone devices, and (b) by a full take of traffic including laptop and desktop devices. While the focus of our work is on mobile devices, many desktop applications display similar behavior as their mobile app versions.

The office network was set up using a *Netgear WNDR4000*[3] router. To enable full featured WiFi and traffic capturing, the alternative firmware *DD-WRT (v3.0-r30016 mega(06/24/16))*[4] was installed. The data was then captured in the *pcap* file format using *tcpdump*[5]. Local traffic was excluded using a Berkeley packet filter.

Name	Interf.	Start	End	Size	Packets
wifi-2,4	eth1	2016-09-05	2016-12-14	12.7 GB	31.5 M
wifi-5	eth2	2016-09-05	2016-12-14	36.8 GB	43.4 M
full	br0	2016-12-16	2017-01-19	30.0 GB	51.5 M

Table 6.3: Captured Data

As outlined in Table 6.3, the data was captured in two phases. In the first phase, traffic was captured on both wireless interfaces with 12.7 GB used over the 2.4 GHz network and 36.8 GB transferred via the 5 GHz network. In the second phase, all traffic was captured, explicitly including wired machines. Local traffic was excluded from all sets. The data was captured in two phases. In the first phase, traffic was captured on both wireless interfaces with 12.7 GB used over the 2.4 GHz network and 36.8 GB transferred via the 5 GHz network over a three months period. In the second phase, all traffic was captured, explicitly including wired machines. Local traffic was excluded from all sets, resulting in 30 GB over a one month period.

Data Processing In the first step, the tool pkt2flow[6] splits up the input file to one pcap per flow before the statistical values of flows are computed. The cluster labeling is implemented in the same way as proposed by Zhang et al. [Zha+13] and Vladutu et al. [VCD16].

Flow Separation While there are many tools to compute flow statistics from *pcap* files, most are trimmed to efficiency and are not extensible in a simple manner. We therefore decided to

[3] `https://www.netgear.com/support/product/WNDR4000.aspx`
[4] `https://www.dd-wrt.com/wiki/index.php/Netgear_WNDR4000`
[5] `http://www.tcpdump.org`
[6] `https://github.com/jonashoechst/pkt2flow`

split up the flow separation and the statistics computation. In the first step, the tool pkt2flow[7] splits up the input file to one pcap per flow. To implement the presented neural autoencoder, the open source library *TensorFlow* [Mar+15] was used.

Statistics Computation To compute the statistical values of flows, a tool was written using Python and the *libpcap* Python library *pypcap*[8]. The feature vector was computed as described in Section 6.1.3. To evaluate different autoencoder configurations quickly, the flow statistics were saved to intermediate files using the Python pickle file format.

Neural Autoencoder and Classification To implement the presented neural autoencoder, the open source library *TensorFlow*[9] is used. The cluster labeling is implemented according to by Zhang et al. [Zha+13] and Vladutu et al. [VCD16]. The labeled data is clustered using the already trained network. The cluster then gets assigned the most frequent label from the previously labeled data. If there are clusters that contain no labels, no cluster label is assigned and the cluster may need further manual inspection.

6.1.5 Experimental Evaluation

In this section, the proposed method is evaluated. To find an optimal configuration, the parameters influencing the classification quality are investigated.

Aggregation Method Two aggregation methods were examined. The cumulative method (cum) is around 15% worse than using the non-cumulative version (noncum), where the flow is separated into multiple segments and the statistical values are computed individually.

Number of Clusters The number of clusters is defined by the number of hidden nodes of our neural autoencoder. A small number of clusters can lead to bad results in terms of discrimination between different subclasses, but the analysis is simpler when smaller numbers of clusters are used. Not every cluster needs to have a meaning, since accidental correlations can also be identified. Hence, a larger number of clusters is not problematic. When a large number of clusters is chosen, small or empty clusters may occur, which can just be discarded. On the other hand, a large number of clusters leads to a more precise distinction between classes. In our parameter scan, we evaluated 10, 15, 20, 30, 40, 60, 80 and 100 clusters. The experiment shows (Fig. 6.4) that a sweet spot can be identified when 60 clusters are used, since the averaged results are not significantly better when more clusters are used.

Scaler The scaler has a major impact on the classification results. While average precision and recall are only at roughly 50% when no scaler is used, the standard scaler improves average precision and recall up to around 60%.

[7] `https://github.com/jonashoechst/pkt2flow`
[8] `https://pypi.python.org/pypi/pypcap`
[9] `https://www.tensorflow.org`

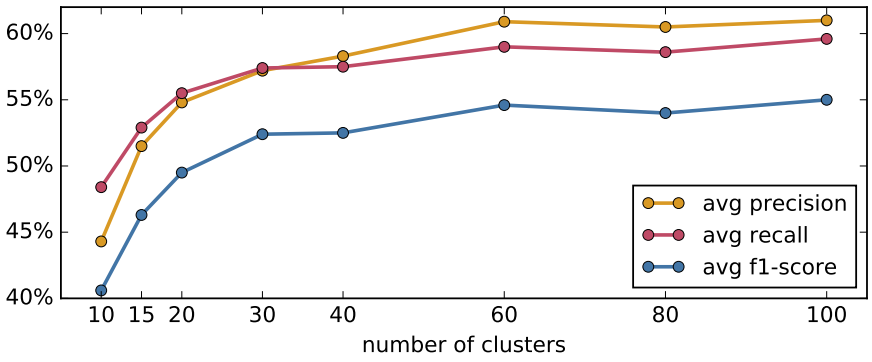


Figure 6.4: Classification quality vs. number of clusters

	precision	*recall*	*F1 score*
videostream	0.47	0.80	0.59
upload	1.00	0.85	0.92
livestream	0.86	0.67	0.75
browsing	0.91	0.50	0.65
download	0.80	0.80	0.80
call	0.87	1.00	0.93
interactive	0.71	0.60	0.65
avg/total	**0.80**	**0.75**	**0.76**

Table 6.4: Classification quality

Classification Quality Table 6.4 shows the obtained classification quality. The neural autoencoder in the configuration with 100 clusters, learning in 30 epochs with a standard scaler based on the full dataset produced the best result. These 100 clusters are mapped by our classifier to our 7 chosen classes. An average precision of 80% and an average recall of 75% are achieved, which results in an F1 score of 76%.

In a QoE sense, live applications should be classified with a high recall. The details of the classification are presented in Table 6.5. The classification of *call* is remarkable, since it has a recall of 100%. The *livestream* class has rather bad values, since it is confused with the *interactive* and the *videostream* class. Since *interactive* subsumes different scenarios including social network browsing, high bandwidth image downloads and video streams can also be part of interactive flows, which can explain this confusion. A subclassification of the *interactive* class should be considered to improve the results.

Execution Time Our runtime experiments were performed on a 2 x 2.26 GHz Intel Xeon quad-core machine. While the generation of flow objects from the pcap file took around 2.20 ms per flow and the computation of the feature vectors took about 1.67 ms per flow, the actual

	video-stream	*upload*	*live-stream*	*brow-sing*	*down-load*	*call*	*inter-active*
videostream	**16**	0	1	0	1	1	1
upload	0	**17**	0	0	0	2	1
livestream	1	0	**6**	0	0	0	2
browsing	8	0	0	**10**	1	0	1
download	3	0	0	1	**16**	0	0
call	0	0	0	0	0	**20**	0
interactive	6	0	0	0	2	0	**12**

Table 6.5: Classification confusion matrix

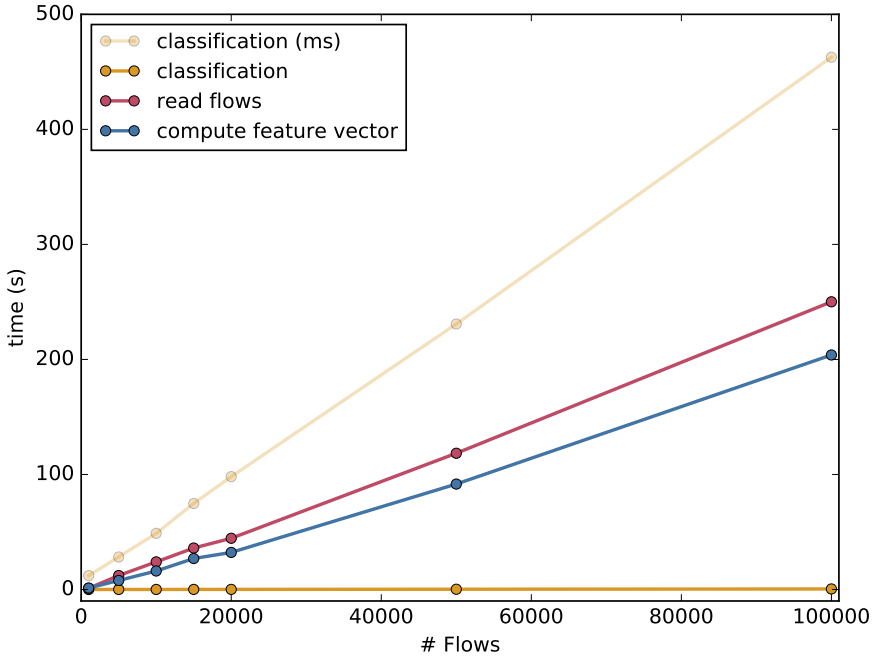


Figure 6.5: Time needed for reading flows, computing feature vector and classification

classification only took 0.006 ms per flow. With a rate of over 200,000 flows per second, our method can be used at the network edges, where access points or mobile devices themselves can classify the traffic and dynamically change connection properties using SDN and SDWN technology to ensure optimal resource usage.

Figure 6.5 shows the time used for reading the flows, computing the feature vector and the actual classification. Since classification itself is very fast, it is additionally depicted on a millisecond scale as the semi-transparent yellow line in the graph. The experiments were performed on a 2 x 2.26 GHz Intel Xeon quad-core machine. The figure shows the computation times for reading flows from disk, computing the feature vector and the actual neural classification. The

first value is only displayed for comparison, since it is not needed in an online classification scenario. The feature vector computation is not yet optimized for performance and currently represents the biggest share of the classification runtime.

The actual classification outperforms both other steps. While the generation of flow objects from the pcap file takes around 2.20 ms per flow and the computation of the feature vectors takes about 1.67 ms per flow, the actual classification only takes 0.006 ms per flow. With a rate of over 200,000 flows per second on the target machine, the results show the feasibility of the method for edge nodes and in particular mobile devices.

6.1.6 Summary

In this section, a novel approach to unsupervised traffic flow classification using statistical properties of flows and clustering based on a neural autoencoder that has been used to cluster traffic flows into downloads, uploads, calls, browsing, videostream, live stream or interactive communication, independent of the particular network protocols used for performing these tasks, was presented. A novel time interval based feature vector construction and a semi-automatic cluster labeling method have facilitated traffic flow classification independent of known traffic classes. Our evaluation using four months of captured traffic has shown that our 7 classes of traffic flows are detected sufficiently fast with an average precision of 80% and an average recall of 75%.

The novel approach to traffic flow classification is a smart solution in the sense of this thesis. The information analysis cost and achievable quality presented in Figure 6.1 on page 184 can be compared with conventional methods, such as port-based classification or deep packet inspection. In the early days of the Internet, certain functionalities were implemented by specific protocols. For example, separate protocols for file transfer (FTP), for interactive connections (Telnet) or voice connections (SIP) were usually also linked to separate port numbers that could be used to classify the connections. However, some of these tasks are now implemented using the WWW to be more available and are thus only recognizable as HTTP traffic, which means that the quality of the classification based on port numbers has greatly decreased. Deep packet inspection uses certain recurring byte sections that lie within the transmitted data to make classification possible. However, the effort required to recognize these patterns for thousands of different protocols is very high and leads, among other things, to delays in recognition. However, as more and more data is encrypted, the quality of this method is also increasingly limited. By restricting the data to statistical information and evaluating it, it is possible to achieve a high quality of the same classes even with changing protocols. The information analysis cost on the other hand is higher than with port-based classification, since only the port number has to be looked up in a table. However, deep packet inspection usually requires complex pattern matching algorithms, which usually have different runtimes due to the memory accesses for different classes. The statistical information required in this approach is collected when the packets are forwarded and does not involve a large information analysis cost; inference using the machine learning model is also feasible in a few milliseconds, as shown above.

6.2 On Dynamic Announcement Intervals in Wireless On-demand Networks

6.2.1 Introduction

Several network protocols rely on nodes broadcasting announcements to other nodes. Examples include service discovery (*Bonjour/ZeroConf, Samba*), routing algorithms (*RIP, OLSR*), and peer-to-peer or delay-tolerant networking (DTN) systems (*Forban*[10], *Serval*[11]). While the traffic generated by periodically sending announcements might be negligible in wired networks with high-speed links, bandwidth in wireless networks, such as 802.11, Bluetooth or various mobile ad hoc networks (MANETs), is precious and limited. For example, spontaneous smartphone networks become more and more important not only by providing pervasive wireless Internet access during large human crowd gatherings, but also during emergency situations or post-disaster recovery [Alo+14].

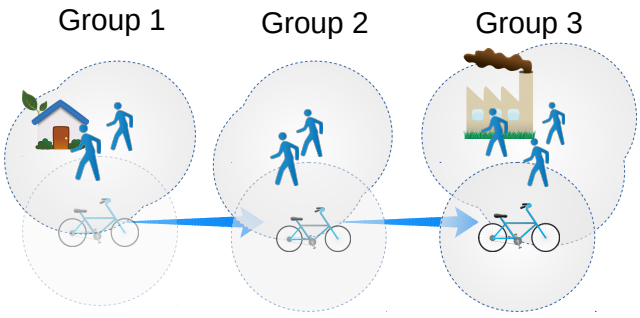


Figure 6.6: Drive-by store-and-forward data exchange.

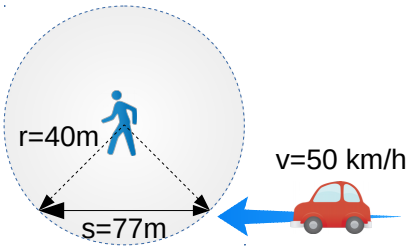


Figure 6.7: Drive-by window of opportunity example.

In an emergency communication scenario, the main goal is to spread messages and files produced at a disaster site fast among reachable nodes. Therefore, data is passed around in an epidemic fashion to as many neighboring peers as possible. Typically, some nodes are more static, such as devices of people trapped in their houses or small emergency camp sites forming islands, while other nodes are on the move (by bike, car, foot), which by passing through these islands act as carrier-pigeons to distribute information further (see Fig. 6.6). These islands have

[10] http://www.foo.be/forban/
[11] http://www.servalproject.org/

a higher density than typical sensor networks. To make optimal use of the short time in case of a drive-by, it is important to find a peer for data exchange very fast. Since any peer can initiate data synchronization, a special treatment of mobility is not necessary. Depending on the used wireless technology, a mobile phone might have an effective range of 14-80 meters to communicate with others. Thus, if we assume a WiFi radius of 40 meters and a static node being 10 meters away from a street, a car driving on the street would be in the WiFi range for about 77 meters (see Fig. 6.7). The car passing by, assuming it moves at about 50 km/h, would have just under 6 seconds for node discovery and exchange of data. This is plenty of time for transferring, for example, two 6 megapixel pictures and setting up connections via a standard 54 Mbps link. Therefore, for the fast moving node, one of the more static peers is sufficient to start a data transfer. Since all information gets replicated in this scenario, the fast moving node does not need to know all possible neighbors. The static node can distribute the data further among its neighbors. Discovering *all* direct peers as fast as possible is neither necessary nor beneficial for the static nodes. Under these assumptions, it is reasonable to use dynamic announcement intervals instead of the typically used static announcement intervals. Furthermore, dynamic announcement intervals require not only less network resources, but also potentially save more battery capacity than static announcement intervals.

In this section, several approaches to realize dynamic announcement strategies that facilitate fast reception from at least one other node while trying to keep the overall communication overhead as low as possible, are presented. Experimental results in terms of performance properties and energy consumption are presented to illustrate the benefits of dynamic announcement intervals in wireless on-demand networks. In particular, the section makes the following contributions:

- Various strategies for realizing dynamic announcement intervals optimized for different network setups are presented.
- An experimental evaluation of all proposed strategies, including static and random announcement strategies, with respect to bandwidth usage, announcement distribution and energy consumption is presented.
- Test environments suited for various topologies, such as large stable networks, islands merging and networks splitting, are investigated.
- The results are directly applicable to local peer-to-peer content distribution systems in emergency scenarios, such as *Forban* and *Serval*.

Parts of this section have been published in Lars Baumgärtner, Pablo Graubner, Jonas Höchst, Anja Klein, and Bernd Freisleben. “Speak Less, Hear Enough: On Dynamic Announcement Intervals in Wireless On-demand Networks.” in: *13th Conference on Wireless On-demand Network Systems and Services (WONS 2017)*. Jackson Hole, USA, Feb. 2017. DOI: 10.1109/WONS.2017.7888768.

6.2.2 Related Work

There are several publications that investigated problems associated with static announcement intervals in various protocols and application scenarios.

Natsheh et al. [Nat+07] proposed a solution based on fuzzy logic to optimize hello messages in dynamic ad-hoc routing. Their work focused on the mesh routing use case, and experiments with a maximum of 35 simulated nodes were presented. Furthermore, Khalaf et al. [KAB10] investigated the broadcast storm problem in mobile ad hoc networks. The authors presented a probabilistic approach to improve the situation in a mesh routing scenario.

Ahmed et al. [ABK15] addressed the problem of beaconing in vehicular ad hoc networks (VANETs). Combinations of controlling a beacon's transmission power, transmission rate, and contention window at the MAC layer were proposed to achieve efficient beacon communication in VANETs. Another approach devoted to improve the problems related to static beaconing intervals in ad hoc networks was presented by Tahar et al. [Tah+16]. Hess et al. [HHO14] investigated peer discovery in mobile opportunistic networks by considering the mobility of nodes.

Peng [Pen15] proposed an adaptive mobility-aware MAC protocol for wireless sensor networks. Apart from optimizing the number of messages, the energy consumption was investigated. Lim et al. [LYD09] presented an approach called RandomCast to improve the energy efficiency of 802.11 ad hoc networks. In this approach, the sender can specify the desired level of overhearing of neighboring traffic, trying to find a balance between energy consumption and routing performance.

Using perfect difference sets for neighbor discovery, Link et al. [Lin+11] presented an energy efficient approach for wireless networks. The authors focused on sensor networks and DTNs with sporadic communication, whereas we focus on networks with higher communication frequencies in local clusters.

Peer-to-peer content distribution is another scenario where announcements are relevant, and a trade-off must be made between central tracker-based peer discovery and distributed peer discovery. Dán et al. [DCC11] presented a hybrid approach that uses individual trackers and a gossip protocol to improve peer discovery. By hopping between swarms and redistributing known peers, efficiency is increased.

Liu et al. [Liu+15] developed a censor-ship resistant delay-tolerant network for message exchange and evaluated it with respect to performance and energy consumption. To avoid energy draining broadcasting with fixed intervals, the authors adopted an approach presented by Zheng et al. [ZHS03] based on asynchronous wake-ups for ad hoc networks. Another delay-tolerant networking system designed specifically for data synchronization in emergency situations was presented by Paul et al. [Pau+16]. While optimizations are proposed to speed up file transfers and syncing, the actual peer discovery was realized by simple broadcasts with fixed announcement intervals.

During an experimental evaluation of *Serval* as a delay-tolerant emergency communication platform, Baumgärtner et al. [Bau+16] found that regular broadcasts used for node discovery or announcements of routing and data storage information especially in networks with many direct peers require high network bandwidth. The study showed that around 2 seconds of announcement delay was the best trade-off between quick peer discovery and conserving energy with the stock implementation made available by the Serval Project.

By exploiting social network characteristics for assisting ad hoc peer discovery, Zhang et al. [Zha+15a] attempted to find optimal beacon probing rates with constant intervals for each group of users. As stated by Wang et al. [WSM07], peer discovery itself can be as energy consuming as making phone calls.

Trifunovic et al. [Tri+11] presented a solution for opportunistic networks of stock mobile devices using 802.11. Since ad hoc mode and Bluetooth pairing does not really work in practice on current mobile devices, open access points and intelligent switching of clients between these access points were used.

While most of the mentioned work is highly specific to the studied use cases, the general picture is that adaptive or dynamic announcement intervals usually outperform static ones, not only with respect to network performance, but also regarding energy consumption. In our scenario, we consider small dense clusters of nodes where a few nodes act as mobile bridges between these islands, in contrast to most sparse sensor networks. Furthermore, most approaches focus on lower layer technologies, whereas our algorithms can be applied on the application layer without operating system support.

6.2.3 Dynamic Announcement Intervals

In this section, several dynamic announcement strategies, the constraints associated with them, and quality properties to evaluate their performance, are presented.

Announcement Strategies

We have developed several novel strategies for realizing dynamic announcement intervals. Each strategy has access to the current announcement delay, the global number of announcements seen at the last observation interval and the current number of unique peers. Our strategies are described in the following:

Static

The *Static* announcement strategy is the basic announcement approach used by most current broadcast protocols. There is a fixed interval defined for every node in which an announcement is sent. This also means that the generated global traffic is growing linearly with the node count. By default, this interval is set to a 2 second delay in our tests, which also is the recommended value for MANET NHDP [CDD11].

Random

In the *Random* strategy, every node chooses a random announcement delay. This delay is a random number between a minimum and a maximum (as described in Section 6.2.3) for every observation interval. The distribution of the random numbers, depending on the network size, heavily influences the performance of this strategy, as well as the duration of the observation interval.

RandomSweet

In this strategy, *Random* is extended. The announcement interval is only set randomly if the current global announcement rate is higher than one announcement per second or less than the minimum number of announcements per second (see Section 6.2.3). Thus, if the network has reached a stable state, this strategy does not change anything and sticks to the last randomized delay for each node. This stabilizes the network if by chance optimal delay combinations are found, at least until nodes join or leave the network.

Step

After every observation interval, the *Step* strategy checks the global announcement count. If the count is higher than one announcement per second, the node's announcement delay is increased by one second. If the count is lower than 0.5 announcements per second, the node's announcement delay is decreased. This leads to gradually narrowing down to a must suitable announcement delay over time.

StepRand

In this strategy, *Step* is extended by adding randomness to each step. While the conditions remain the same as in *Step*, a random value between 0 and 0.5 seconds is added or subtracted to the announcement delay.

MaxFirst

MaxFirst is a rather defensive strategy: whenever a high global announcement rate is detected (more than one announcement per second), the node's announcement interval is set to the observation interval, i.e., the maximum possible announcement delay is tried first, hence the name. Then, if less than 0.5 global announcements per second are present, the strategy decreases the announcement delay by one second per iteration, until the local minimum of 0.5 seconds is reached. Thus, a very low announcement frequency is favored, which should be beneficial in larger or fast growing networks.

MinFirst

MinFirst reverses *MaxFirst*, and thus is an aggressive announcement strategy. Whenever less than 0.5 announcements per second are detected globally, the announcement delay is set to the local minimum of 0.5 seconds. Otherwise, the announcement delay is increased by one second per iteration, until the observation interval is reached. This strategy supports scenarios where most of the time only very few peers are in direct vicinity of each other.

Unsteady

In the *Unsteady* strategy, each announcement delay is computed only on the basis of the number of unique peers known by a node and not on the global announcement rate like in the other algorithms. The goal is to reach a global rate of one announcement per second. Looking at the current peer count, an announcement interval is computed to complement the announcement intervals of the other nodes. Using this method, the strategy should be able to adapt to new situations as fast as defined by the observation interval.

Constraints

To guarantee that a node can be discovered, we define an *observation delay*, with the same value for all nodes. This is the time between re-evaluation and before another change in the announcement frequency can happen. Each node has to announce itself at least once per observation interval. All nodes must set the announcement delay after the observation delay is over. This enables a better comparability between the announcement strategies.

The *observation delay* is set to 20 seconds in all our experiments, since the baseline for static announcements is 2 seconds. Therefore, it is reasonable to re-evaluate the situation after 10 standard announcements. The higher the delay, the longer it takes for the network to adapt to new situations. A very short delay in conjunction with the premise that each node should at least send one announcement per interval leads to higher loads, especially with higher node numbers. Thus, a delay of 20 seconds ensures that within this interval *all* peers in the direct neighborhood are discovered.

Quality Properties

To evaluate and compare different strategies for dynamic announcement intervals, universally applicable quality properties must be defined. Our main goal is to globally have one announcement per second at any given time, not less, but also not much more to conserve resources. This goal is motivated by the drive-by scenario described in Section 6.2.1, in which 10% of the window of opportunity would be used for peer discovery under this assumption.

Global Announcement Rate

The *Global Announcement Rate* is measured by counting the announcements per second. This parameter is the main optimization goal for our algorithms, since it is directly correlated with the bandwidth used for peer discovery.

Global Announcement Gaps

The *Global Announcement Gaps* are measured by the time periods between two announcements. The *Global Announcement Gaps* are important to observe, since they reveal how long a new peer needs until it receives an announcement from the rest of the network. Although this value is roughly the inverse of the *Global Announcement Rate*, its distribution can reveal other aspects, as observed in our experiments.

Adaptation Rate

The *Adaptation Rate* represents the time needed for an announcement strategy to adapt to a new situation. It describes the situation that all nodes are started at the same time, and defines the moment when no significant change in the number of announcements is recognizable.

6.2.4 Implementation

In this section, implementation issues of our announcement strategies and the network using them are discussed.

Mesher

To investigate dynamic announcement intervals, we extended a simple broadcast service to provide easily exchangeable announcement algorithms for peer discovery. *Mesher*[12] is a simple local chat written in Google's *Go* language by one of the authors, and therefore is easily extensible. It utilizes broadcast packets for neighbor discovery and for exchanging public chat messages. *Mesher* uses a static announcement interval of 2 seconds in its default configuration, and thus the network traffic is growing linearly with the node count. Each announcement contains the elliptic curve public key of the sending node, the services provided by the node, 512 bytes random data to simulate database states and a cryptographic signature, resulting in 642 bytes per broadcast packet. Other protocols might use larger or smaller announcement packets, depending on the type of state that is broadcasted.

Dynamic Interval Computation

To evaluate various interval computation methods including dynamic changes, the corresponding algorithms needed to be easily exchangeable. Therefore, we decided to implement the algorithms using an embedded *JavaScript* engine, and defined an interface to hand over useful information to access it in the main *Go* binary:

- `get_announce_count()`
- `get_and_reset_announce_count()`

[12] *Mesher*, available online: https://github.com/gh0st42/mesher

Listing 6.1: Basic layout of the announcement strategies

```
var observation_interval = 20000;
var total_count = 0;
var min_delay = 500;

set_announce_delay(2000);
for(;;) {
  sleep(observation_interval);
  var cur_count = get_and_reset_announce_count();
  var cur_delay = get_announce_delay();

  // call scheduler and set new delay there
  scheduler(cur_count, cur_delay);
}
```

- `get_peer_count()`
- `get_announce_delay()`

After analyzing the provided values, the algorithms are able to set a new announcement interval using `set_announce_delay(Int)`.

Announcement Strategies in Mesher

For all announcement strategies, we used the same template (see Listing 6.1) in *JavaScript* where one specific function is responsible for computing any changes. Each strategy gets the current announcement delay and the global number of announcements seen in the last observation interval. This setup proved to be perfect for rapid prototyping of new algorithms without recompilation or modifications of the main binary.

6.2.5 Experimental Evaluation

To evaluate our announcement strategies, several setups were used, including emulations with many nodes as well as physical machines connected over various network links.

Network Emulation

For network emulation, we selected the Common Open Research Emulator[13] (*CORE*), which is scriptable using *Python* and in this way allows versatile creation of experimental configurations. This system uses Linux and lightweight virtualization to provide a networking testbed for unmodified, regular Linux binaries. All announcement strategies are evaluated under four different network scenarios described below:

[13] CORE: http://www.nrl.navy.mil/itd/ncs/products/core

Centralized Network

In the *Centralized Network* configuration, all nodes are connected centrally and hence are located in the same collision domain. This setup is similar to a classic network hub or a local ad hoc wireless network in the sense that each node can directly communicate with all of its adjacent peers. As long as the network is not oversaturated, every node gets the announcements of every other node.

Growing Network

In the *Growing Network* configuration, nodes are added periodically to the network. Ideally, the announcement strategies should adapt to the new situation fast and down-regulate their announcement counts. Each second, a new node joins the network, and adaptation is required to maintain optimal resource usage.

Merging Network

In the *Merging Network* configuration, two equally sized *Central Networks* merge at a fixed point in time, doubling their size instantaneously. Using this configuration, adaptation rates for abruptly changing network configurations can be observed.

Splitting Network

In the *Splitting Network* configuration, the network is split in two halves at a fixed point in time. By creating two independent networks, the announcement strategies need to react fast to satisfy the defined quality properties and avoid prolonged periods of silence between announcements.

Physical Testbed

To evaluate the proposed announcement strategies under realistic conditions, a physical testbed was created. It consists of several Raspberry Pi 3 Model B[14] single-board computers, running under the vendor-provided Debian-Linux-based Raspbian[15] operating system. This platform is comparable to mobile phones in terms of energy consumption and therefore allows us to obtain realistic energy and power consumption measurements when evaluating the announcement strategies.

We set up eight Raspberry Pis as network participants, as well as an additional Raspberry Pi as a system under test (SUT). The energy consumption of the SUT was measured using an *Odroid Smart Power* measurement device, an external power meter. The data points were logged at 5 Hz to another device, in order to prevent disruption of the measurement.

[14] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[15] https://www.raspbian.org

Experimental Results

In this section, the announcement strategies described in Section 6.2.3 are evaluated using the network configurations of Subsection 6.2.5. Based on the quality properties of Section 6.2.3, the strategies are compared to each other.

To test our strategies, the centralized network configuration was evaluated with different node counts. For each of the eight announcement strategies, the tests were performed using 2, 5, 10, 25, 50, 100 and 200 nodes, resulting in 56 configurations. These configurations were each executed using two nodes starting mechanisms: a) the batch node start, in which all nodes were started randomly in the observation interval window; b) the delayed node start, where a node was added every second, resulting in a linearly growing network.

In addition, two dynamic network configurations were used: *Split*, where the central network was split in two halves, and *Merge*, where two equally sized networks were joined. Summing up the different configurations, 224 independent experiments were performed.

Basic Capabilities

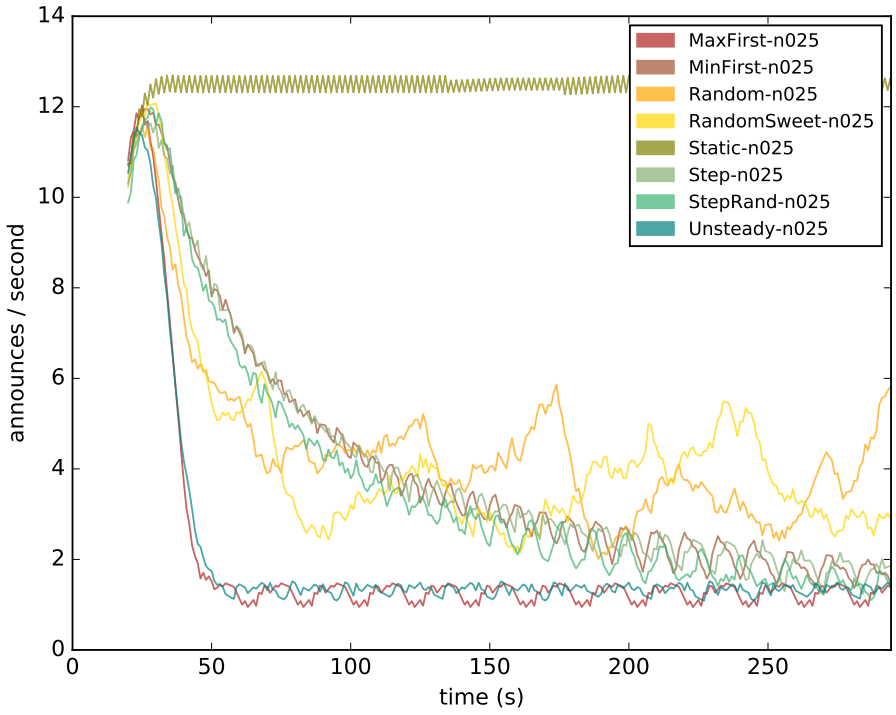


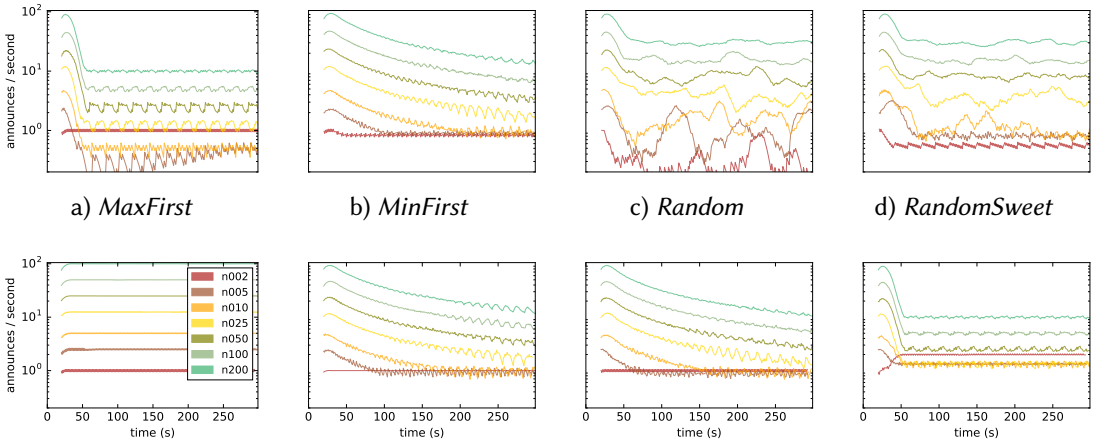
Figure 6.8: Announcements/second in a static network of 25 nodes.

In Fig. 6.8, the announcement rate for all strategies in a static network with 25 nodes is visualized. The strategies share the same observation interval, and therefore the first 20 seconds are the same, since they also start with the same announcement interval of 2 seconds. The *Static* strategy preserves this announcement interval, and the globally generated traffic remains the same for the whole experiment.

Unsteady and *MaxFirst* show very low announcement rates in this network configuration. *Unsteady* uses the node count (see Sec. 6.2.3) and computes its maximum announcement delay, which in this case is greater than the observation delay and sets this maximum. *MaxFirst* jumps to the maximum possible announcement delay, since the observed announcement count is high. Since the situation does not change, both algorithms stick to their decision in future observations. This similarity changes for lower node counts. Considering Figure 6.9a, *MaxFirst* sets the same very low announcement rates in the beginning, which leads to low global announcement rates and finally to big gaps between each two announcements. *Unsteady* (Fig. 6.9h) compensates this problem and starts with higher announcement rates in smaller networks.

MinFirst and *Step* also behave similarly, since the down steps are implemented the same way. Both algorithms extend their announcement delay by 1 second, starting at a delay of 2 seconds. *StepAndRand* also is in the same group and only differs from *Step* by adding a random value with a maximum of 0.5 seconds. All three algorithms achieve the goal of a less saturated network and also approach the same minimum as *MaxFirst* and *Unsteady*.

In this network configuration, *RandomSweet* as well as *Random* show a similar behavior. The announcement rate drops directly after the initial observation, but stays higher than for the other strategies that achieve a low announce rate after around 200 seconds. To get similar results as, for example, *MaxFirst*, all nodes would need to pick a pretty high delay by chance, and the more nodes in the network, the more unlikely it is that all nodes do this in the same observation interval.



a) *MaxFirst* b) *MinFirst* c) *Random* d) *RandomSweet*

e) *Static* f) *Step* g) *StepRand* h) *Unsteady*

Figure 6.9: Comparison: announcements produced by the proposed strategies in different static network configurations.

Bandwidth Savings

A major goal for using dynamic announce intervals is the reduction of bandwidth in such protocols. Table 6.6 shows the announcement rates of the proposed strategies compared to the

Name \ # Nodes	2	5	10	25	50
Static	291	732	1460	3658	7296
Random	34,4%	47,0%	37,0%	37,9%	37,3%
RandSweet	58,1%	41,7%	29,0%	35,6%	37,7%
Step	101,7%	45,4%	35,2%	33,2%	33,4%
StepRand	99,7%	42,5%	32,5%	30,1%	30,2%
MaxFirst	99,0%	21,2%	17,1%	17,0%	17,1%
MinFirst	84,9%	44,3%	34,7%	33,3%	33,5%
Unsteady	188,7%	56,8%	32,5%	17,7%	17,1%

Table 6.6: Announcements of the strategies compared.

static announcement strategy. For this table, the announcements sent by one node in the batch node start is used. This number also includes the observation delay in which all strategies follow the static behavior.

All non-static strategies converge for growing node counts. *Step*, *StepRand* and *MinFirst* use around a third of the number of announcements compared to *Static*. *MaxFirst* and *Unsteady* take advantage of their fast adaptation rate and are able to save around 80% of the announcements. This means that only one fifth of the bandwidth is used without sacrificing any comfort or usability of the protocol.

Table 6.6 also shows that the proposed strategies benefit the most from their dynamic behavior for networks with 2 to 10 nodes. After that, only minor improvements can be achieved. The announcement rate of *Static* can be altered easily by hand and could therefore also reach the goal of a lower global announcement rate for big networks, but would then lose the ability to perform good in small networks without manual interaction on each node.

Unsteady uses more bandwidth than *Static* for a minimal network. This allows fast discovery of new peers in an existing network and addresses the real-world problems described in Figure 6.7. *Random* and *RandomSweet* have a lower total announcement count in small networks. This shows that these strategies are inferior in terms of discovery times. The remaining *Step*-based strategies show satisfactory results in small and bigger networks in terms of bandwidth usage, but take a longer time to reach an optimal resource usage.

Adaptation Rate

Unsteady and *MaxFirst* have a very high adaptation rate, since they set their final announcement delay after the first observation interval for all static network configurations, as presented in Figure 6.9h. *MaxFirst* is able to achieve fast adaptation rates for big networks, while *MinFirst* is able to achieve this in small networks, as a result of their designs. A disadvantage of *MaxFirst*

is shown in Figure 6.9a: For small networks, the announcement rate also drops to the minimum in the first place, so discovery may be worsened.

The adaptation rate of the *Step*-based algorithms depend on the number of nodes. As outlined in Figure 6.9f, in a network of 5 nodes around 70 seconds and in a network of 10 nodes around 150 seconds are needed to fully adapt.

In Figure 6.10, a splitting network configuration with 10 nodes is presented. The *Step*-based strategies reach their target announcement rate immediately. In *RandomSweet* and *Unsteady*, new announcement rates are visible after about 30 seconds. Both strategies reach announcement rates as in the united, central network. This understanding only slightly differs in the merging network: The *Step* based algorithms need longer, while *Unsteady* and *MaxFirst* adapt in the observation interval.

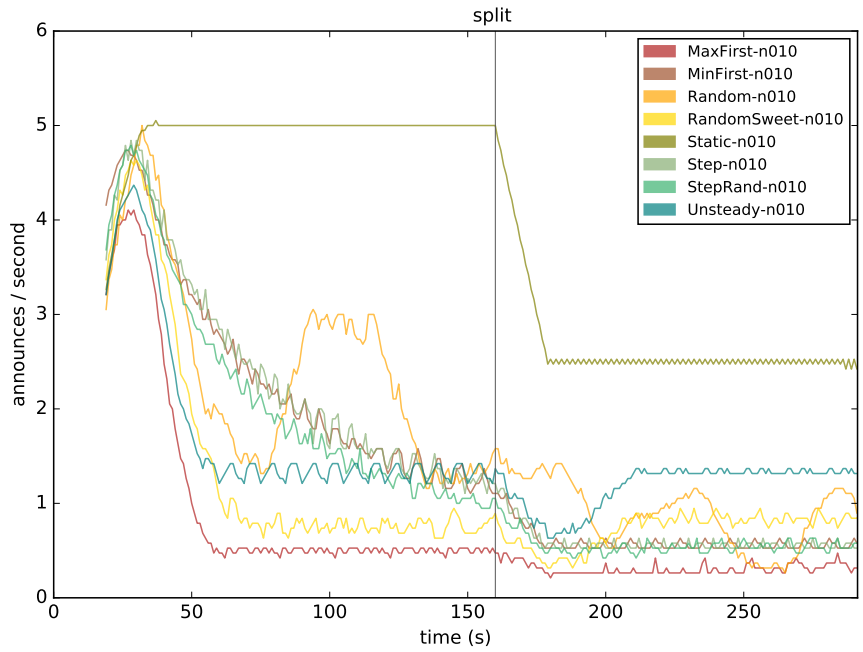


Figure 6.10: Splitting network configuration with 10 nodes.

The observed adaptation rates are also valid for the merging network configuration: *MaxFirst* and *Unsteady* adapt in a 30-seconds window, while the *Step* strategies take a longer time. For the network of 5 nodes, the *Step* strategies also achieve an adaptation rate of around 40 seconds. Especially in small networks, this rate is important, since the announcement gaps are compensated quickly.

Figure 6.11 shows a delayed start of 100 nodes, with one node starting per second. Compared to *Static*, the proposed algorithms are able to keep the announcement rates low. Since every node announces using the default interval for the first 20 seconds, the announcement rate grows even in the very agile *Unsteady* and *MaxFirst* strategies. Immediately after all nodes are spawned, the algorithms are able to adapt to the situation.

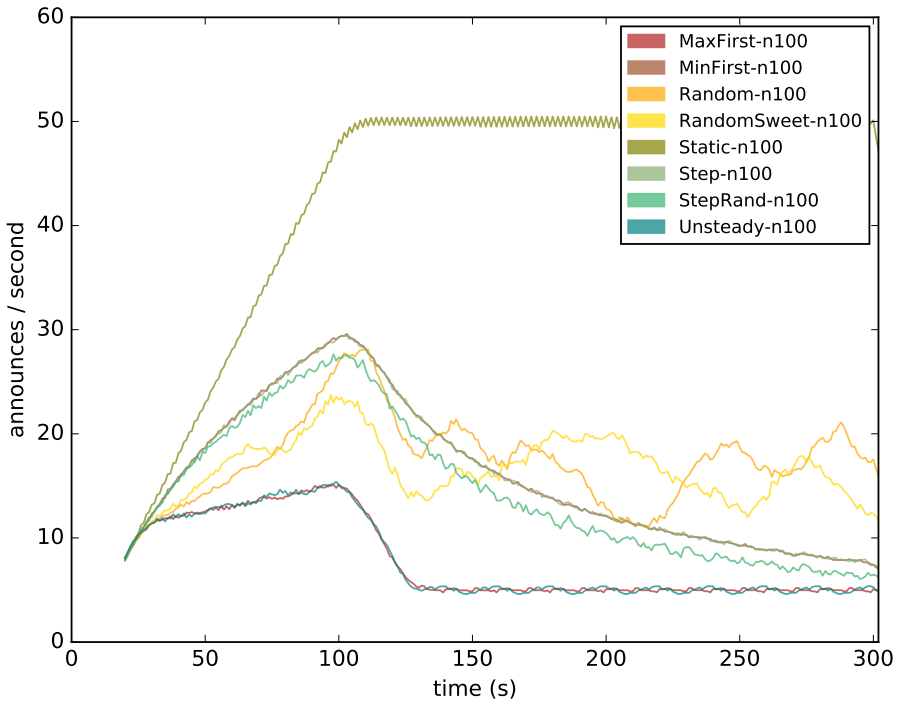


Figure 6.11: A growing network with 100 nodes.

Announcement Gaps

Figure 6.12 shows a violin plot of the global announcement gaps for a static network with 10 nodes. The mean gap correlates with the global announcement rate, and so does the variance. Having this in mind, the perceptions of Subsection 6.2.5 are backed by this plot. Although *MaxFirst* does not have the highest announcement gap, it produces a relatively high percentage of longer gaps, while all other strategies only have a low number of outliers in this area. This is also the case for a network of 5 nodes. Yet larger network configurations do not show the same characteristics. This behavior can be ascribed to the observations made in the previous section.

What stands out is that compared to *Static*, all strategies perform worse with respect to the maximum announcement gap. This can be put in perspective by examining the upper quartile: For all algorithms except for *MaxFirst*, the upper quartiles of the announcement gaps are below 2 seconds.

Energy Consumption

Our initial assumption was that a reduced number of announcements would reduce the consumed energy proportionally. We evaluated this assumption in a wireless network of 9 ARM-based nodes as described in Section 6.2.5. In these experiments, each node acted as sender and receiver simultaneously. One node (system under test - SUT) was connected to an external power meter (ODROID SmartPower), which logged the power and energy consumption of the

node at a 5 Hz rate. Additionally, we ran every experiment with two different network interface configurations, with a different idle power consumption each: ad hoc mode (P_{idle}=1.37 W) and managed mode (P_{idle}=1.45 W).

To measure the higher end of the power consumption, two additional announcement strategies sending announcements at a high rate are introduced: *Static05* and *Static01*, with 2 and 10 announcements per second, respectively.

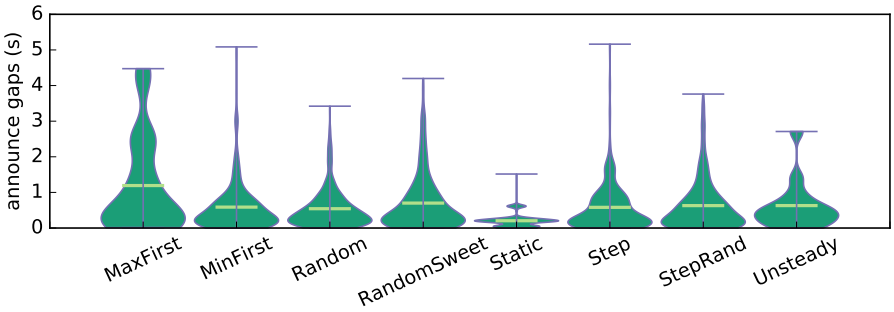


Figure 6.12: *Announcement Gaps* in a static network of 10 nodes.

To compute the energy consumed by our software, the average idle power is subtracted from the measured power in the given 300 seconds measurement interval:

$$E := \int_0^{300} P_{measured}(t)\,\mathrm{d}t - 300 * P_{idle} \tag{6.1}$$

In the physical testbed with 9 nodes, the default *Static* strategy uses 1.99 mWh. *Static05* and *Static01* use 11.97 mWh and 32.52 mWh for their announcements, respectively. Based on these numbers, a correlation between the number of announcements (sent and received) and the consumed energy is found and presented in Table 6.7.

While the correlation between the number of announcements and the energy consumption is reasonable for large numbers of announcements, this correlation is not substantial for lower numbers of announcements. The general trend seems to be correct (correlation coefficient $r = 0.985$), since all proposed strategies need less energy than *Static*. In contrast, there are examples in which this correlation seems to be vice versa, e.g., when comparing *MaxFirst* and *RandomSweet*.

To summarize, the energy measurements of our experiments show that for high numbers of announcements the energy consumption is increased. Side-effects of the programming language, as well as the relatively low energy impact of the announcements of *Mesher* disturb the energy measurements. Nevertheless, a general trend is clearly evident.

6.2.6 Summary

In this section, it was shown that without relying on application-specific properties, optimizations for network protocols relying on announcements can be achieved. We have compared eight dynamic announcement strategies, including a standard static announcement strategy

Name	# Ann.	E (mWh)	rel. Ann.	rel. E	ratio
Static	1323	1.99	1.00	1.00	1.00
Static05	5404	11.97	4.08	6,00	1.47
Static01	29342	32.52	22.18	16.31	0.74
MaxFirst	256	1.17	0.19	0.59	3.04
MinFirst	473	1.26	0.36	0.63	3.04
Random	434	1.34	0.33	0.67	2.04
RandomSweet	342	0.73	0.26	0,37	1.42
Step	495	1.20	0.37	0.60	1.61
StepRand	460	1.12	0.35	0.56	1.61
Unsteady	514	1.38	0.39	0.69	1.78

Table 6.7: Correlation of energy consumption and announcements in a physical testbed of 9 nodes.

and a random announcement strategy. While a random announcement strategy might preserve more bandwidth than a static announcement strategy, it has negative side-effects compared to the other proposed announcement strategies. By dynamically changing the announcement interval and depending on the number of nodes involved, we were able to reduce the bandwidth required for announcements by more than 80% compared to a static announcement strategy. Nevertheless, our requirements of fast discovery of at least one node are still met. The evaluation of the proposed announcement strategies in terms of energy consumption show that announcements do effect battery lifetimes and are thus worth to be reduced.

The algorithms for dynamic announcement intervals presented here are a smart solution in the sense of this thesis. The information analysis cost and achievable quality shown in Figure 6.1 on page 184 can be classified using two technical metrics. The QoS refers directly to the expected delay in service discovery. Conventionally, static announcement intervals are used, thus the quality is highly variable. In a small group, it takes a comparatively long time for a node to make a discovery, while in a large group, the announcements can generate cross-talk. The information analysis cost is mainly expressed by the unnecessarily sent announcements. The solution presented here allows an efficient discovery in scenarios and groups of different sizes with a comparatively low information analysis cost overhead.

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

6.3.1 Introduction

Smartphones have become our daily mobile companions to provide wireless access to communication, information, and entertainment services. Since a large amount of data is not downloaded in advance but streamed on demand via the Internet, seamless connectivity using both Wi-Fi and cellular interfaces is desirable. The mobility of smartphone users leads to the problem of deciding when to use which wireless connection. Most smartphones use Wi-Fi as their default interface, since many cellular data plans will be throttled after exceeding a certain limit. The decision when to perform vertical handovers is often based on the Wi-Fi received signal strength indicator (RSSI) and timeouts for the transmission of packets. However, if a user is leaving a location while listening to a music stream or watching a streamed video, the established Wi-Fi connection eventually becomes unavailable and the streaming process stops. The mobile operating system detects the connection loss some time after the connection is lost. Finally, the application needs to reestablish the connection.

In this section, a novel approach to predict Wi-Fi connection loss before the connection breaks to perform seamless vertical Wi-Fi/cellular handovers, is presented. Our approach relies on data collected by multiple smartphone sensors (e.g., Wi-Fi RSSI, acceleration, compass, step counter, air pressure) to predict Wi-Fi connection loss and uses Multipath TCP (MPTCP) to dynamically switch between different wireless connectivity modes. We train a random forest classifier and an artificial neural network on roughly 20 GB of sensor data collected by five smartphone users over a period of three months. The trained models are efficiently executed on smartphones and reliably predict Wi-Fi connection loss 15 seconds ahead of time, with a precision of up to 0.97 and a recall of up to 0.98. Furthermore, we present results of four DASH video streaming experiments that run on an Android smartphone and make use of available Wi-Fi/cellular networks. The neural network predictions for Wi-Fi connection loss are used to establish MPTCP subflows on the cellular link. Our experiments show that the proposed approach provides seamless wireless connectivity, improves quality of experience by increasing mean opinion scores (MOS) from 2.7 to up to 3.8 for certain scenarios, and requires up to 50% less cellular data compared to handover approaches without Wi-Fi connection loss predictions. The data set, analysis scripts, experimental logs, and the mobile app developed in this paper are publicly available[16]. To summarize, we present:

- a novel approach to predict Wi-Fi connection loss for performing seamless vertical handovers,
- a neural network to learn and predict Wi-Fi connection loss based on a novel combination of smartphone sensors,
- a vertical handover method that uses MPTCP for switching between wireless connection modes at runtime,

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

- an implementation on off-the-shelf smartphones to demonstrate its performance in real-world scenarios; our results show significant improvements in terms of Quality of Experience and the amount of cellular data consumed.

Parts of this section have been published in Jonas Höchst, Artur Sterz, Alexander Frömmgen, Denny Stohr, Ralf Steinmetz, and Bernd Freisleben. "Learning Wi-Fi Connection Loss Predictions for Seamless Vertical Handovers Using Multipath TCP." in: *2019 IEEE 44th Conference on Local Computer Networks (LCN 2019)*. **Best Paper Award**. Osnabrück, Germany, Oct. 2019. DOI: 10.1109/LCN44214.2019.8990753. URL: https://umr-ds.github.io/seamcon.

6.3.2 Related Work

Predicting Wi-Fi Connection Loss

Several approaches to predict Wi-Fi connection loss for performing handovers have been proposed in the literature [ABG14]. Nasser et al. [NGA07] use neural networks to predict Wi-Fi connection loss events based on RSSI. Horich et al. [HJG07] use a fuzzy logic controller (FLC) for making decisions about performing handovers, where the parameters for the FLC are learned using a neural network.

Lin et al. [LWL08] propose to use standard Wi-Fi connection properties and a neural network to predict Wi-Fi connection loss. Monsour et al. [MEO17] use a combination of user velocity and the Allan variance of the RSSI to predict Wi-Fi connection loss, and use PMIPV6 to manage the predicted Wi-Fi connection loss. Khan et al. [Kha+17] propose a fuzzy logic system to predict Wi-Fi loss events based on various parameters, such as delay, jitter, bit error rate, packet loss, communication cost, response time, and network load.

These approaches are limited to information of wireless connections, which may be helpful to create metrics for Wi-Fi quality, but is not always the best information for predicting Wi-Fi connection loss. In contrast, our approach considers information from a wide range of smartphone sensors that indicate the usage context, leading to high-quality predictions.

Other approaches incorporate the mobility of the users [NN08] or higher level features like social group affiliation, time-of-day, and average duration a user spends in a particular network [Wan+11].

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

Performing Vertical Handovers

There are extensions to the traditional Internet Protocol that allow users to keep a session alive when (vertical) handovers are performed [Per10]. These approaches are based on home and foreign agents that forward traffic for the mobile host. Although they are around for a long time, mobile IP is not supported widely. Ma et al. [Ma+04] propose a vertical handover

method based on the Stream Control Transmission Protocol. While the proposed method is network-independent and thus does not require home and foreign agents, traditional TCP-based applications cannot benefit from the advancements. MPTCP is a TCP extension supporting multiple subflows for a single TCP connection [For+13]. MPTCP improves throughput and reliability in data center and mobile environments [Rai+12; Che+13]. Paasch et al. [Paa+12] evaluate MPTCP as a vertical handover mechanism. The authors propose three MPTCP modes for handover scenarios, namely *Full*, *Backup*, and *Single-Path Mode*. The first two modes maintain subflows on all interfaces, while the *Single-Path Mode* exploits the break-before-make design of MPTCP. Pluntke et al. [PEK11] use MPTCP as a vertical handover mechanism to shift connections between cellular and WiFi connectivity and finally to save energy. De Coninck and Bonaventure [DB17] futher improve the handover by speeding up packet retransmissions after the cellular subflow is established.

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

6.3.3 Conceptual Overview

Figure 6.13 shows the components of our approach and the workflow. First, raw sensor data is collected by a mobile app developed for our work and uploaded to a server for further processing. The raw data is appropriately preprocessed and enriched with additional higher level features. The resulting data is then used to train and evaluate a random forest classifier and different neural network architectures. The data preprocessing operations as well as the trained models are transpiled to Java code and integrated into the mobile app on the smartphone, which in turn makes online predictions for Wi-Fi connection loss 15 seconds ahead of time. Based on these predictions, vertical Wi-Fi/cellular handover is performed using MPTCP. We explain the main steps of our approach in more detail below. Neural network model building and Wi-Fi connection loss predictions for performing MPTCP handovers on a smartphone are discussed in Sections 6.3.4 and 6.3.5, respectively.

Smartphone Sensors

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

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

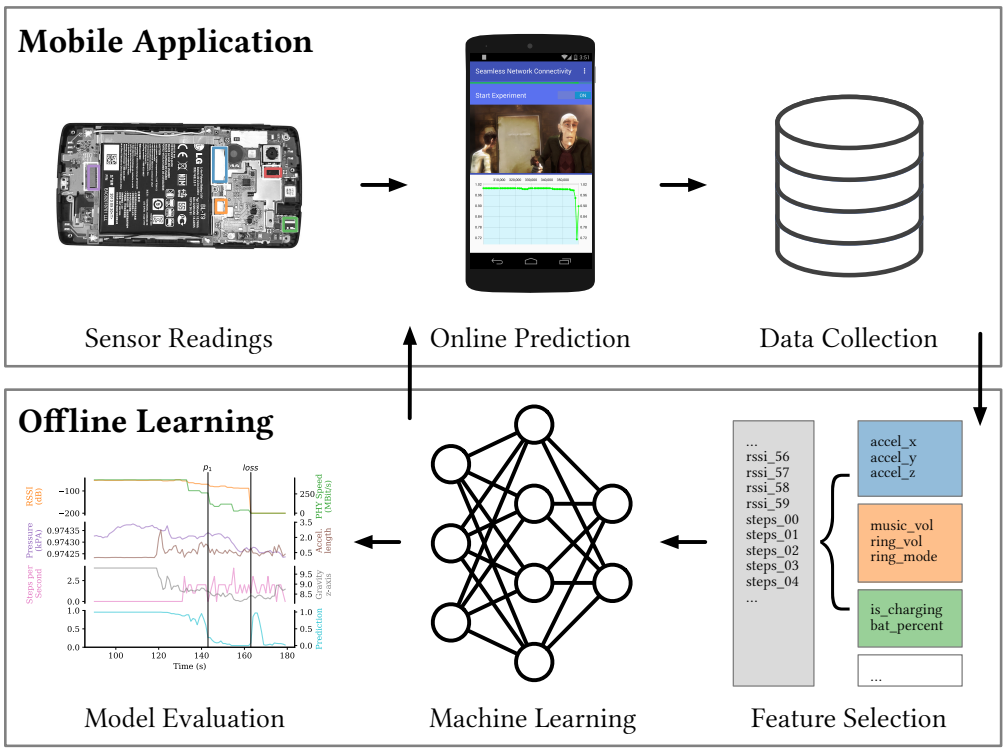


Figure 6.13: Mobile application and offline learning.

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

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

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

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

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

Sensor Data Preprocessing

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

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

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

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

Precision and Recall

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

6.3.4 Learning Wi-Fi Loss Predictions

In this section, our novel data-driven approach to predict Wi-Fi connection loss is presented.

Data Set

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

Training and test set Machine learning methods require separate data sets for training and testing to verify the generalization abilities of the trained models. We investigated different ways of building training and test sets: (a) we randomly split the available samples into, e.g., 70% training and 30% test data, and (b) we split by users, to learn and test with different users.

Feature Vectors

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

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

Reduced Feature Vector Many of the sensors, like linear acceleration and gyroscope, described in Section 6.3.3 share underlying features due to their physical properties. The number of sensors can be reduced by leaving aside these sensors. For the *Reduced Feature Vector*, we used the following sensors: Atmospheric pressure: $delta$; Linear acceleration: $length$; Step counter: $delta$; Power: $is\ charging$; Gravity: z; Wi-Fi: $frequency$, $speed$, $RSSI$.

Sensor Data Example

Figure 6.14 shows an example of several sensor data values collected by a smartphone. The figure shows the computed ground truth and a prediction probability value of a neural network based on the *Full Feature Vector*, i.e., a probability value $< 50\%$ means that a Wi-Fi connection loss is predicted and vice versa. The graphical representation of the sensor values shows that no obvious correlation between one of the sensors and the prediction ground truth exists. Nevertheless, each of the sensors shows some information that could be useful. For example, the atmospheric pressure sensor rises from $t = 100$ to $t = 115$, which could be caused by changing the floor in order to leave the building or by a changing ventilation. In combination with the step counter delta, the first option is more likely, also resulting in a higher likelihood for a Wi-Fi connection loss. Another example is the gravity sensor's z axis that reports about

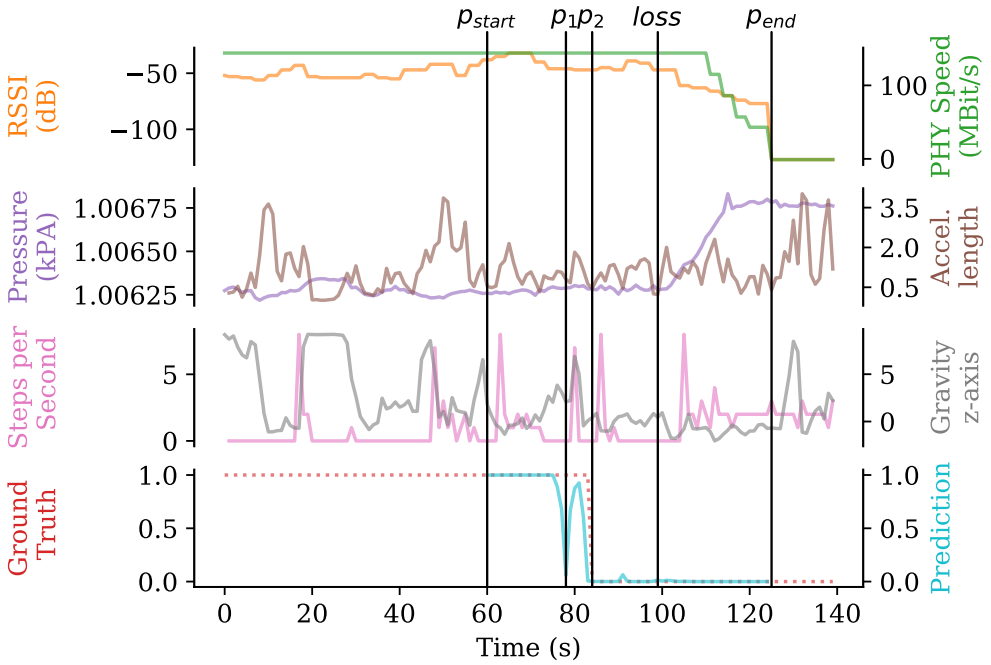


Figure 6.14: Different sensors leading to an early (p_1) and an ideal (p_2) prediction of Wi-Fi connection loss, based on a trained model with randomly split data.

9.81 for the time period from $t = 20$ until $t = 35$, which together with the linear acceleration sensor is a good sign for laying flat on a table. This again reduces the likelihood of a Wi-Fi connection loss event.

For the neural network shown on the bottom in Figure 6.14, a 60 seconds observation window has to be filled before the first prediction is performed at p_{start}. The classification ends at p_{end}, since the operating system reports that Wi-Fi is unavailable. Since Wi-Fi becomes unavailable at *loss*, the ground truth is 0 from p_2 ongoing, matching the 15 seconds prediction window. The neural network classifier matches the ground truth quite well, with the exception of p_1, where the classifier predicts the loss slightly too early. This example shows that the combination of sensors available on today's smartphones can lead to an effective prediction of Wi-Fi connection loss.

Machine Learning Results

This section presents results of training different methods with the data to predict Wi-Fi connection loss: (a) a random forest classifier [LW02], and (b) a multi-layer neural network. In particular, we use the MLPClassifier and RandomForest implementations of scikit-learn [Ped+11].

Random forest Since random forest learning depends on equally distributed samples, the data is down-sampled accordingly to match this criterion. The random forest consists of 10 random trees, learned using the Gini criterion. The overall performance of the random

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

Table 6.8: *Reduced Feature Vector*, randomly split data, different learners and configurations.

forest is satisfactory, since all values are greater than 0.97. However, the precision of the Wi-Fi connection loss class was not very high (0.86), ultimately resulting in triggering early or unnecessary handovers.

RSSI-only neural network Another basic learning approach is to limit the learner to only use the timeseries of RSSI values, as presented in Section 6.3.2. During our experiments, different configurations of the neural network were evaluated. The overall performance is comparable to the performance presented in the related work. The classification quality of the Wi-Fi connection loss class did not exceed an F_1-score of 0.95.

Random Data Split The results for neural networks learned with randomly split data depend on the neural network architectures. Table 6.8 provides an overview of different classifier approaches with the *Reduced Feature Vector*. Classifier *NN 1* consists of 100 hidden neurons, *NN 2* of (300, 200, 100) neurons, and *NN 3* of 5 hidden layers containing (400, 400, 400, 400, 400) neurons. All results were achieved using 70% of the data set exclusively for learning and the remaining 30% for testing. In our experiments, *NN 1* can reach a classification quality comparable to the random forest classifier. The F_1-score of the Wi-Fi connection loss class reaches up to 0.94, with either a high precision or a high recall, but never both. In general, the negative class, representing stable Wi-Fi connections, is predicted well by all tested neural network classifiers. The experiments show that neural networks can reach both high precision and high recall in the positive Wi-Fi connection loss class.

The results presented in Table 6.8 show that *NN 2* and *NN 3* provide reasonably good performance for both precision and recall in the Wi-Fi connection loss class. Even the neural network *NN 2* consisting of three layers shows significant improvements compared to the flat neural network discussed in the previous paragraph. It reaches an F_1-score of 0.96 with slightly lower recall or precision.

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

User-based Data Split When testing for previously unseen users, the precision of the loss worsens in our prediction. With 0.93, 0.92, and 0.79 precision in the Wi-Fi loss class, the *Reduced*

Feature Vector generalizes better compared to the *Full Feature Vector* resulting in 0.91, 0.72, and 0.68 precision.

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

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

6.3.5 Experimental Evaluation

As presented in Section 6.3.4, the learned neural network models reliably predict Wi-Fi connection loss with a high precision and recall. To show the usefulness of these results, we evaluated the performance when performing handovers in real-world mobile usage scenarios. In the following, we present a seamless Wi-Fi/cellular handover during DASH video streaming sessions.

Seamless Network Connectivity App

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

Sensor data collection and preprocessing The sensor readings described in Section 6.3.3 are cached in memory and written periodically to a local SQLite database on the smartphone. When a run ends, the database is uploaded to a server. To execute the neural network on the smartphone, the sensor values are preprocessed similarly to the offline learning process. The mean, variance, and the observation window determined offline are used.

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

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

streaming session while the video is playing to the server including stalls, playback bit rates, quality adaptations, and buffer levels for later evaluation.

MPTCP handovers We use the Wi-Fi connection loss prediction to trigger the cellular subflow establishment for MPTCP *before* the Wi-Fi connection is lost. We implemented our approach on top of the MPTCP kernel implementation for Android[17] and the *MultipathControl*[18] app of De Coninck et al. [De +16]. Furthermore, the video server uses MPTCP version 0.92 with the redundant scheduler and the fullmesh path manager enabled.

Experimental Setup

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

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

The following set of routes is chosen to evaluate scenarios in which Wi-Fi connection losses can occur. Figure 6.15 shows the room plan of the university building where the tests were performed.

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

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

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

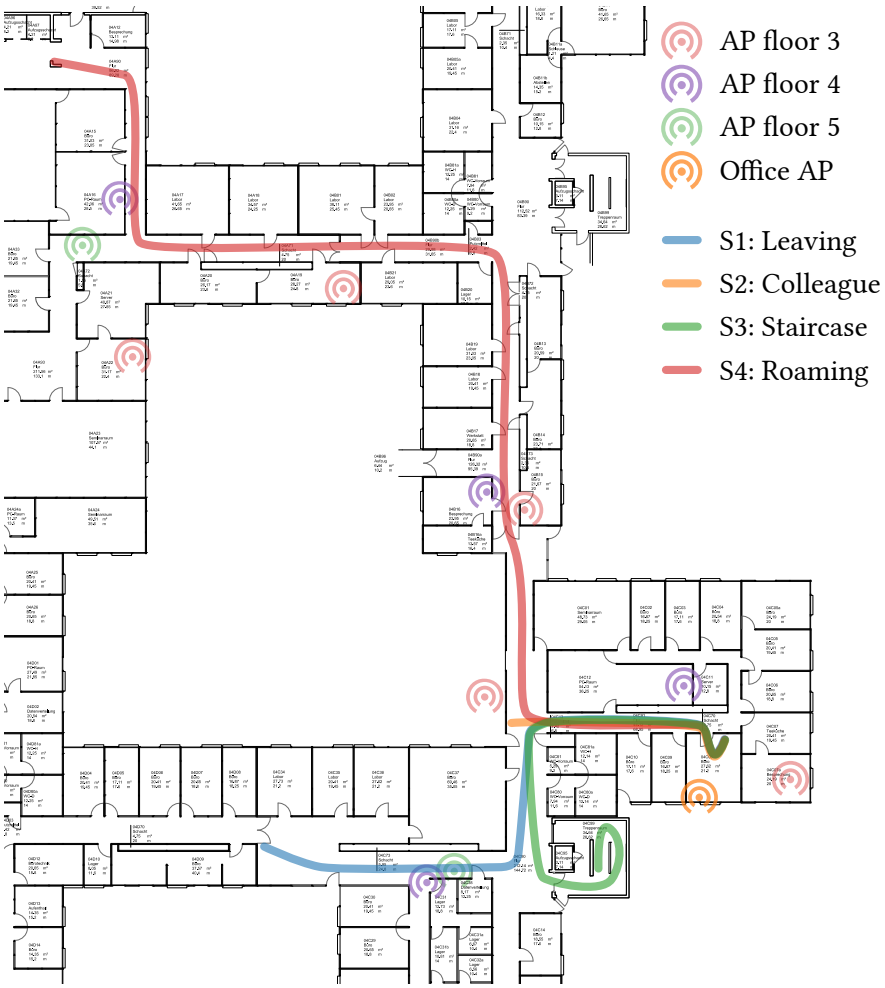


Figure 6.15: Map with Wi-Fi APs and scenarios routes.

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

Scenario 4: Wi-Fi roaming support Starting in the office, the device is connected to the university network. The tester leaves after 120 seconds and heads towards the other end of the building, roaming between multiple possible Wi-Fi APs shown in Figure 6.15. The tester stays near the exit for 10 seconds and then walks back the same route. This scenario is created to further investigate the support of roaming gaps in corporate wireless networks where roaming might be available, but is not sufficient to achieve a high QoE.

Measuring Quality of Experience

The DASH video streaming technology is widely available, used by many vendors, and evaluated well. To measure the perceived QoE, several technical values are captured that are used to compute mean opinion scores (MOS) [Sto+16], as discussed below.

Direct Measurements

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

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

QoE Metrics

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

$$MOS_{stall} = 3.5 \times e^{-(0.15 \times L + 0.19) \times N} + 1.5 \tag{6.2}$$

$$MOS_{quality} = 0.003 \times e^{0.064 \times t \times 100} + 2.498 \tag{6.3}$$

$$MOS_{combined} = \frac{MOS_{stall} + MOS_{quality}}{2} \tag{6.4}$$

Finally, Stohr et al. [Sto+16] propose the average MOS, denoted as $MOS_{combined}$ (Equation (3)), denoting a total user perception not only depending on stalling or quality adaptations. We use $MOS_{combined}$ to evaluate QoE.

(a) Scenario 1: Leaving

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

(b) Scenario 2: Colleague

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

(c) Scenario 3: Staircase

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

(d) Scenario 4: Wi-Fi Roaming

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

Table 6.9: Overview of Experimental Results

QoE Experimental Results

In Table 6.9, the overall results of the performed tests are presented, namely the number of stalling events (# St.) and the average duration of a stalling event (∅ St.), the number of adaptations (# A.), the relative time in the highest playback quality (HQ), and the average transmitted data (∅ TD).

Scenario 1 As shown in Table 6.9a, the *Stock* tests performed worst with 3 stalling events in total and an average stalling duration of about 1.5 seconds, while neither *MPTCP* nor *Seamless* tests did show any stalling events, which is a significant improvement compared to the stock tests. The amount of transferred data over cellular is high in the *MPTCP* test and low in the *Stock* test. *Seamless* results are between these two tests, thus saving cellular data compared to MPTCP, while still avoiding stallings. The results of these tests show that our prediction can avoid the handover gap completely.

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

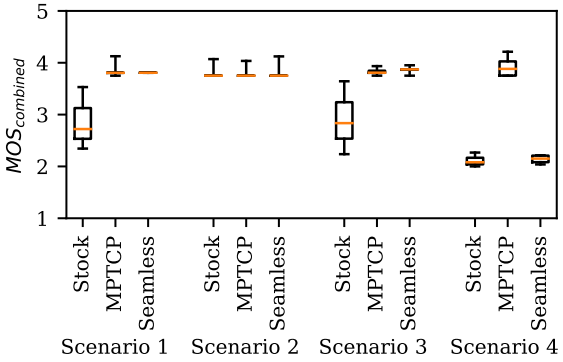


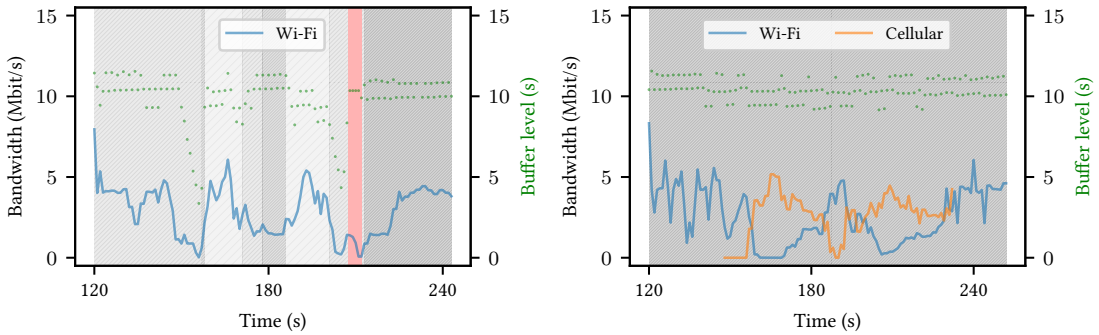
Figure 6.16: $MOS_{combined}$ values grouped to connectivity modes and scenarios.

Apart from improvements of these technical values, our approach improves QoE for users, as expressed in the $MOS_{combined}$. Figure 6.16 shows the $MOS_{combined}$ on the y-axis and the different connectivity modes on the x-axis, grouped by scenario. For the stock tests, the $MOS_{combined}$ is between about 2.5 (poor) and 3.5 (fair), indicating that the playback is not totally unsatisfactory, but far away from a great experience. *Seamless*, on the other hand, achieves a $MOS_{combined}$ of almost 4, indicating a good QoE, as high as in *MPTCP* tests.

Scenario 2 As shown in Table 6.9b, all tests are comparable for all metrics, showing that our approach does not introduce any negative effects in already good situations. The transferred amount of data over cellular in *Seamless* is about as high as in the *MPTCP* tests. This is because the classifier predicts a Wi-Fi connection loss due to the movement of the smartphone and thus switches to the cellular network, even though this is not necessary.

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

Scenario 3 As shown in Table 6.9c, the stock tests performed worst with 3 stallings and an average stalling time of about 2 seconds. Additionally, with 49 adaptations and only 80% of the time at the highest achieved quality, the stock tests perform badly. *MPTCP* and *Seamless* do not stall at all. With 28 and 30 adaptations and 85% of the time at the highest achieved quality, the results of our approach are as good as in the *MPTCP* tests, again showing significant improvements over the stock implementation. The data usage over cellular shows the same behavior as in *Scenario 1*.



a) Stock Android b) Seamless

Figure 6.17: *Stock* and *Seamless* in *Scenario 3*

Figures 6.17a and 6.17b show bandwidth and buffer level for *Scenario 3*. In the stock tests, the maximum distance is shown in the used bandwidth around seconds 150 and 210. *Seamless* improves this situation and establishes a cellular connection in a timely manner resulting in no stallings. $MOS_{combined}$ during the stock tests shows again a relatively bad QoE with about 2.5 to 3.5 compared to the high MOS values of about 4 during MPTCP and tests using our approach.

Scenario 4 The results of the stock tests in Table 6.9d indicate that Android does not handle *Scenario 4* well. The video stalls 18 times and for about 15 seconds on average. The video quality adapts 42 times in total and stays only for 53% of the time at the highest achieved quality. MPTCP, on the other hand, handles *Scenario 4* very well with no stallings, few quality adaptations, and 86% of the time at the highest achieved quality.

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

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

$MOS_{combined}$ during the *Seamless* and stock tests is comparably bad with a value of about 2, while MPTCP still reaches a MOS of about 4. Nevertheless, our approach reaches a slightly higher QoE than stock Android, as shown in Figure 6.16.

Overhead Analysis

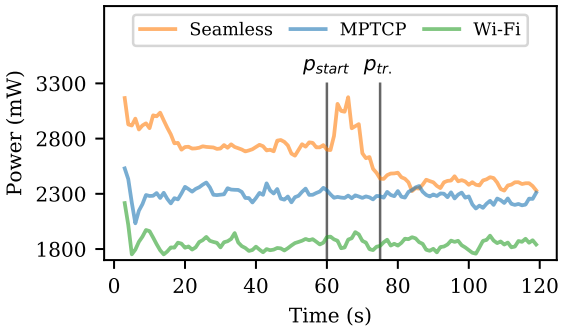


Figure 6.18: Power Consumption

Executing a neural network on a mobile device introduces overhead in terms of CPU and memory usage, which is mainly reflected in power consumption. To evaluate this overhead, power measurements were performed on a Nexus 5 smartphone. To avoid interferences from the battery, we removed the battery from the device, soldered wires to the charge controller and put it back into the Nexus 5. Then, we powered and measured the device using a Monsoon

High Voltage Power Monitor[19] with a sample rate of 5 kHz and a resolution of 286 μA. The voltage was set to 4.2 V, which corresponds to about 92% battery capacity.

Since the measured device was not mobile anymore due to the wired power setup, we performed a series of independent tests that are illustrated in Figure 6.18. First, we performed a baseline measurement using only Wi-Fi, where neither the on-device prediction nor the sensor logging was enabled (denoted as *Wi-Fi*). The second experiment additionally incorporated cellular connectivity (marked as *MPTCP*). During the last experiment, the Wi-Fi connection loss prediction (*Seamless*) was measured. In all tests, the video was streamed on the device as in all other tests (see Sec. 6.3.5. Furthermore, the MPTCP kernel was used in all experiments. The x-axis of Figure 6.18 shows the time in seconds and the y-axis denotes the consumed power in mW. All tests were executed for 120 seconds, whereas in the prediction test the first 60 seconds were used for filling the observation window and the prediction started at p_{start}.

The average power consumption during the Wi-Fi tests is about 1856 mW with a standard deviation of 145 mW. Enabling LTE on the device, the average power consumption increases by 18.9% to about 2289 mW and a standard deviation of 143 mW.

The power consumption using our prediction approach depends on the current state. During the first 60 seconds, LTE was enabled. Here, the average consumption is about 2792 mW, which is about 22% higher than during the MPTCP test. Shortly after p_{start}, no Wi-Fi connection loss is predicted, thus the LTE is switched off, which causes the device to change routing tables, turn the LTE device off, and MPTCP has to reschedule its subflows. Doing this increases the power consumption to an average of 2843 mW. At $p_{tr.}$, the transition from MPTCP over Wi-Fi and LTE to Wi-Fi only has finished, so that the power usage decreases to about 2420 mW. Compared to the consumption during the Wi-Fi tests, this is an overhead of 30.6%. Compared to the MPTCP test, the prediction introduces an overhead of 5.7%. In total, to retain the same high QoE, our approach introduces an overhead of 15.7% on average (2649 mW, 300 mW standard deviation), including all three phases of the *Seamless* tests.

Considering the 9660 mWh battery of the Nexus 5, the video could be streamed for about 4.22 hours continuously using MPTCP. Incorporating our on-device prediction, the time decreases to 3.65 hours, which are about 34 minutes less watching time. This is a reasonable overhead for executing a neural network in an Android app, and it can be further improved. For example, Graubner et al. [Gra+18b] showed that certain computations can be performed in an energy-efficient manner by executing them on a mobile device's sensor hub or Wi-Fi chip.

6.3.6 Summary

In this section, a novel data-driven approach to predict Wi-Fi connection loss to perform seamless vertical Wi-Fi/cellular handovers, was presented. The approach is based on sensors available in today's smartphones and uses MPTCP to dynamically switch between different wireless connectivity modes. We demonstrated that our trained neural networks reliably predict Wi-Fi connection loss 15 seconds ahead of time when users move around, with a precision of up to 0.97 and a recall of up to 0.98. Furthermore, we illustrated the benefits of our Wi-Fi connection loss prediction approach with an MPTCP video streaming application. We showed

[19] https://www.msoon.com/

that our predictions improve the QoE mean opinion score from 2.7 to up to 3.8 for certain scenarios, while reducing the required cellular data usage by up to 50% compared to traditional MPTCP approaches, with a negligible power consumption overhead.

The approach presented here for seamless Wi-Fi/cellular handovers is a smart solution in the sense of this thesis. In this section, achievable quality as shown in Figure 6.1 on page 184 refers in particularly to the QoE perceived by the user. The information analysis cost has two characteristics. On the one hand, there is the computational cost, since the sensor values have to be evaluated continuously and the WiFi loss prediction is carried out. On the other hand, the redundant transmissions made via MPTCP contribute to this cost.

The Mean Opinion Score (MOS) can be used as a quality metric. In the case study, different scenarios are defined for which different quality improvements were measured. In a scenario without Wi-Fi connection loss, no quality improvement was measured (control group). In two other scenarios where the Wi-Fi connection was lost and a MOS of 2.7 was previously achieved, the deployed smart system allows an increase to 3.8. In order to quantify the information analysis cost in this case study, the overhead analysis presented in Section 6.3.5 can be used, since a full system power measurement provides a good estimate of the overall cost. Following the energy measurements, an overhead of 30.6% compared to Stock Android is introduced by the approach. Another conventional approach to seamless handovers is pure MPTCP, where parallel connections are established over cellular networks and Wi-Fi.

Our seamless handover approach achieves similar quality to the MPTCP approach, but the connection loss prediction generates an overhead of 15.7%. Following these two quality and cost metrics, it must be noted that the approach is close to the ineffective category as introduced in Figure 3.1. Another metric can be used for evaluation, namely the cellular data consumption, which is limited for most users on a monthly basis. The data consumption is in all scenarios was lower compared to the MPTCP approach, in some cases it was reduced by 78.46%. Furthermore, the implementation of Wi-Fi loss prediction in this work is not optimized for energy consumption, although enormous savings can be expected here in particular.

7 Conclusion

This chapter concludes the thesis and outlines areas of future work with respect to the areas of our contributions.

7.1 Summary

In this thesis, smart systems were presented in three different areas: environmental monitoring, adaptive disruption-tolerant networking, and transitional wireless networking. An overarching categorization was presented that can be used to evaluate smart systems based on achievable quality and information analysis cost.

In the area of smart distributed sensing, in particular in the field of environmental monitoring, the main research question was how the flexibility of single-board computers could be facilitated for smart distributed sensing. In particular, the following contributions were presented:

- A novel approach for configuring single-board computer operating system images, called PIMOD, was presented.
- A novel, open-source software for reliable VHF radio tracking of small animals in their wildlife habitat, called tRackIT OS, was presented.
- A multi-sensor approach that combines ultrasonic audio recordings, automatic radio telemetry, and video camera recordings in a single modular unit, called BatRack, was presented.
- Bird@Edge, a novel Edge AI system for recognizing bird species in audio recordings to support real-time biodiversity monitoring was presented.

In the area of adaptive disruption-tolerant networking and in particular in opportunistic function execution, improvements in terms of improved QoS and/or reduced information analysis cost, were presented:

- An in-depth experimental evaluation of the delay-tolerant aspects of Serval for various network setups and usage patterns was presented.
- Opportunistic named functions as a novel approach to operate ICN-DTNs during emergencies were presented.

- A novel framework for offloading computational workflows in opportunistic networks, with two addressing modes, workers publishing their capabilities and available resources, a worker assignment algorithm, appropriate error handling, and network cleanup to reduce network load, was presented.
- A novel open source DTN implementation, called DTN7, of the recently released Bundle Protocol BP7, written in the Go programming language, was presented.
- A novel approach to support *Programmable Disruption-tolerant Networking* by allowing network operators to program a node's routing behavior based on context information, without requiring knowledge of the router's interior workings, was presented.
- A novel, freely available and open source modem firmware for LoRa-enabled MCUs, called *rf95modem* was presented, a device-to-device LoRa chat application for iOS, Android, and laptop/desktop computers was created, and an integration of LoRa into the disruption-tolerant networking software DTN7 was presented.

The insights from smart distributed sensing and smart adaptive disruption-tolerant networking were applied to transitional wireless networks and improved achievable quality and/or reduced information analysis cost. In particular, the following contributions were presented:

- A novel approach to unsupervised traffic flow classification using statistical properties of flows and clustering based on a neural autoencoder independent of the particular network protocols was proposed.
- Several novel approaches to realize dynamic announcement intervals for the announcements of groups that kept communication overhead as low as possible were presented.
- A novel data-driven approach to predict Wi-Fi connection loss to perform seamless vertical Wi-Fi/cellular handovers was presented.

7.2 Future Work

This section presents future work for smart distributed sensing in adaptive wireless networks. Future work will be considered in the three areas of environmental monitoring, adaptive disruption-tolerant networking, and transitional wireless networking.

7.2.1 Smart Environmental Monitoring

Environmental monitoring is a quite old field, considering that nature observations go back hundreds of years. With the advent of modern technologies, not only the observation tools, but also the observation itself and evaluation process are becoming digital. To further increase the achievable quality, future work should be done in the following areas:

An interesting area of future work is to extend PIMOD to enable reproducible image builds, which can also support reproducible research in itself. This is currently limited by non-deterministic side effects, such as creating temporary and logging files, as well as time stamps contained in the image file system.

For reliable VHF radio tracking of small animals, calculating exact bearings can be challenging, since signals are affected by multiple factors, such as vegetation, topology of the surrounding area, humidity, and rainfall. While bearings can be directly calculated based on a simple model, higher quality can be achieved by using data of multiple stations and further context information, such as a topology model and/or a calibration for the specific area of operation. Finally, the continuous preparation and further processing of the collected data is the next major task in creating a user-friendly and widely applicable animal tracking system for generating ecological knowledge.

For presented edge AI approach for bird species recognition, self-supervised learning could be used to leverage the vast amount of unlabeled data and to improve the recognition quality on the target domain. Furthermore, continual and federated learning of machine learning models at the edge are interesting future research topics.

Probably the biggest and most challenging task in environmental monitoring will be the consolidation and integration of diverse data sources and very large amounts of data. Techniques such as network processing or edge AI, which have already been partially applied in this thesis, can help to cope with these data volumes.

7.2.2 Smart Adaptive Disruption-tolerant Networking

Adaptive, disruption-tolerant networking itself is a niche technology, judging by its prevalence. However, there are very convincing use cases in which the technology can offer real added value. For the efficient application of the technology in these use cases, some future work can be done:

For some of the presented contributions, i.e., Serval and DTN7, mobility simulations should be carried out, preferably with real world movement patterns gathered from past events. A simulator for the effective evaluation of DTN software should be created that allows researchers to evaluate different approaches quickly and comparably. Also, further applications for delay tolerant networks, such as remote medical support in regions with sparse populations or during a disaster, should be discussed.

Regarding function execution as presented in opportunistic named functions and OPPLOAD, exploring incentive mechanisms for function execution to perform functions locally for a global benefit, possibly by leveraging a game-theoretic approach, are of interest. Incorporating further network or social knowledge could improve overall performance in both opportunistic approaches and should be investigated further.

There are several areas for future work in the presented DTN7 implementation. For example, the bundle protocol does not define any kind of security or privacy mechanisms, although optional extension exist. This opens the field of DTN-related security and privacy research based on DTN7. Furthermore, for sensor networks or deployments in rural areas, DTN7's energy consumption should be evaluated. Finally, new convergence layers based on emerging radio technologies, such as LoRa or mmWave communication, could be developed.

Regarding the presented programmable DTN approach, implementing a system that allows updating or replacing routing algorithms at runtime would reduce unnecessary downtimes

and further reduce development and deployment hurdles. Also, allowing a centralized entity to reconfigure an entire DTN deployment would make the administration and monitoring of DTN nodes more flexible.

In the presented LoRa device-to-device approach, to efficiently use LoRa and its limited bandwidth in crisis scenarios, a frequency plan for users and first responders should be created. Such a plan can be integrated into the emergency communication app, and the plan could be presented to the user.

7.2.3 Smart Transitional Networking

Smart Transitional Networking is an exciting new field whose improvements can go far beyond incremental improvements within individual protocols. The contributions presented in this thesis can be followed by the following future work:

Regarding unsupervised network traffic flow classification, there are multiple areas of future work, such as (a) using deep and especially stacked autoencoders [Vin+10] to improve the mapping of classes to clusters, (b) replacing the *SoftMax* classification method by other methods to improve the classification, and (c) training the network with subflows of varying lengths to use the approach for nearly real-time classification after observing only a few seconds of the packet stream.

For the presented dynamic announcement intervals, so far the algorithms only have access to information like the number of announcements received in the last observation interval or the number of currently known peers. By giving the strategies more information, further optimizations might be possible. Furthermore, a dynamic observation interval could be implemented, to allow even faster adaptation to new situations. In addition, the proposed announcement strategies should be tested in real-world applications where more computations are needed to generate the announcements, e.g., sending database state by using hashing functions, or transmitting routing tables in a mesh network. Finally, the proposed announcement strategies should be implemented in existing software platforms such as *Serval* or *DTN7* where they could make a difference in real world scenarios.

In the area of Wi-Fi loss predictions and seamless vertical handovers, there are several areas of future work. While the contextual sensors used in the approach support high-quality predictions, other more domain-specific sensors might be useful to predict, e.g., Wi-Fi overloads. It would also be interesting to learn predictions for user/access point combinations. To deploy predictors efficiently on off-the-shelf smartphones, lightweight neural networks on dedicated processing engines should be considered. Finally, in addition to Wi-Fi connection loss prediction, Wi-Fi connection regain prediction is an interesting area for future research.

A particular challenge for the entire field will be the application of the results shown in existing networks and products. There are various obstacles, such as restricted networks or inaccessible, unchangeable, proprietary systems and devices, which cannot be used for the application of these novel approaches, or even hinder their use. Future work in this area would therefore be to investigate how systems can be designed from the ground up to achieve inherent upgradeability in order to support future developments.

List of Figures

List of Tables

Bibliography

[ABG14] Atiq Ahmed, Leila Merghem Boulahia, and Dominique Gaiti. "Enabling Vertical Handover Decisions in Heterogeneous Wireless Networks: A State-of-the-Art and A Classification." in: *IEEE Communications Surveys Tutorials* 16.2 (2014), pp. 776–811. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.082713.00141 (cit. on p. 213).

[ABK15] Syed Hassan Ahmed, Safdar Hussain Bouk, and Dongkyun Kim. "Adaptive Beaconing Schemes in VANETs: Hybrid Approach." in: *2015 International Conference on Information Networking (ICOIN)*. IEEE. 2015, pp. 340–345 (cit. on p. 198).

[Ahn+18] Sanghong Ahn, Joohyung Lee, Sangdon Park, SH Shah Newaz, and Jun Kyun Choi. "Competitive Partial Computation Offloading for Maximizing Energy Efficiency in Mobile Cloud Computing." in: *IEEE Access* 6 (2018), pp. 899–912 (cit. on p. 114).

[Ahr+08] Jeff Ahrenholz, Claudiu Danilov, Thomas R Henderson, and Jae H Kim. "CORE: A Real-time Network Emulator." in: *Military Communications Conference*. IEEE. 2008, pp. 1–7 (cit. on pp. 109, 120, 135, 147).

[All18] LoRa Alliance. "LoRaWAN Regional Parameters v1.0.3." in: *LoRa Alliance: Fremont, CA, USA* (2018) (cit. on p. 166).

[All77] Jonathan Allen. "Short Term Spectral Analysis, Synthesis, and Modification by Discrete Fourier Transform." in: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 25.3 (1977), pp. 235–238 (cit. on p. 46).

[Alo+14] Gianluca Aloi, Marco Di Felice, Valeria Loscrì, Pasquale Pace, and Giuseppe Ruggeri. "Spontaneous Smartphone Networks as User-Centric Solution for the Future Internet." in: *IEEE Communications Magazine* 52.12 (2014), pp. 26–33 (cit. on p. 196).

[Alt+19] Bastian Alt, Markus Weckesser, Christian Becker, Matthias Hollick, Sounak Kar, Anja Klein, Robin Klose, Roland Kluge, Heinz Koeppl, Boris Koldehofe, et al. "Transitions: A Protocol-Independent View of the Future Internet." in: *Proceedings of the IEEE* 107.4 (2019), pp. 835–846 (cit. on pp. 2, 12).

[Alt20] Steven Alter. "Making Sense of Smartness in the Context of Smart Devices and Smart Systems." in: *Information Systems Frontiers* 22.2 (2020), pp. 381–393 (cit. on pp. 10, 11).

[Ana+16] Carlos Anastasiades, Tobias Schmid, Jürg Weber, and Torsten Braun. "Information-centric Content Retrieval for Delay-tolerant Networks." in: *Computer Networks* 107 (2016), pp. 194–207 (cit. on p. 96).

[Asc+19] Fernando Ascensão, Andreas Kindel, Fernanda Zimmermann Teixeira, Rafael Barrientos, Marcello D'Amico, Luís Borda-de-Água, and Henrique M Pereira. "Beware that the Lack of Wildlife Mortality Records can Mask a Serious Impact of Linear Infrastructures." in: *Global Ecology and Conservation* 19 (2019), e00661 (cit. on p. 40).

[Ash+16] Muhammad Ashar, Hirohiko Suwa, Yutaka Arakawa, and Keiichi Yasumoto. "Priority Medical Image Delivery Using DTN for Healthcare Workers in Volcanic Emergency." in: *Scientific Phone Apps and Mobile Devices* 2.1 (2016), p. 9 (cit. on p. 97).

[Aug+16] Aloÿs Augustin, Jiazi Yi, Thomas Clausen, and William Townsley. "A Study of LoRa: Long Range & Low Power Networks for the Internet of Things." in: *Sensors* 16.9 (2016), p. 1466 (cit. on p. 156).

[Ban+19] Aman Bansal, Apoorv Gupta, Deepak Kr. Sharma, and Varshika" Gambhir. "IICAR - Inheritance Inspired Context-aware Routing Protocol for Opportunistic Networks." in: *Journal of Ambient Intelligence and Humanized Computing* 10.6 (June 2019), pp. 2235–2253. ISSN: 1868-5145. DOI: 10.1007/s12652-018-0815-2 (cit. on p. 141).

[Bau+15] Lars Baumgärtner, Jonas Höchst, Matthias Leinweber, and Bernd Freisleben. "How to Misuse SMTP over TLS: A Study of the (In) Security of Email Server Communication." in: *Trustcom/BigDataSE/ISPA, 2015 IEEE*. vol. 1. IEEE. 2015, pp. 287–294. DOI: 10.1109/Trustcom.2015.386 (cit. on p. 7).

[Bau+16] Lars Baumgärtner, Paul Gardner-Stephen, Pablo Graubner, Jeremy Lakeman, Jonas Höchst, Patrick Lampe, Nils Schmidt, Stefan Schulz, Artur Sterz, and Bernd Freisleben. "An Experimental Evaluation of Delay-Tolerant Networking with Serval." in: *2016 IEEE Global Humanitarian Technology Conference (GHTC)*. Seattle, USA, Oct. 2016. DOI: 10.1109/GHTC.2016.7857262 (cit. on pp. ix, 7, 82, 96, 116, 118, 135, 160, 198).

[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: *13th Conference on Wireless On-demand Network Systems and Services (WONS 2017)*. Jackson Hole, USA, Feb. 2017. DOI: 10.1109/WONS.2017.7888768 (cit. on pp. ix, 6, 120, 197).

[Bau+18] Lars Baumgärtner, Alvar Penning, Patrick Lampe, Björn Richerzhagen, Ralf Steinmetz, and Bernd Freisleben. "Environmental Monitoring Using Low-Cost Hardware and Infrastructureless Wireless Communication." in: *2018 IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE. 2018, pp. 1–8 (cit. on pp. 26, 113, 156).

[Bau+19] Lars Baumgärtner, Patrick Lampe, Jonas Höchst, Ragnar Mogk, Artur Sterz, Pascal Weisenburger, Mira Mezini, and Bernd Freisleben. "Smart Street Lights and Mobile Citizen Apps for Resilient Communication in a Digital City." in: *2019 IEEE Global Humanitarian Technology Conference (GHTC 2019)*. Seattle, USA, Oct. 2019. DOI: 10.1109/GHTC46095.2019.9033134 (cit. on pp. 6, 26).

[Bau+20] Lars Baumgärtner, Alexandra Dmitrienko, Bernd Freisleben, Alexander Gruler, Jonas Höchst, Joshua Kühlberg, Mira Mezini, Richard Mitev, Markus Miettinen, Anel Muhamedagic, Thien Duc Nguyen, Alvar Penning, Dermot Pustelnik, Filipp Roos, Ahmad-Reza Sadeghi, Michael Schwarz, and Christian Uhl. "Mind the GAP: Security & Privacy Risks of Contact Tracing Apps." in: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications*

(TrustCom). vol. 1. IEEE. Dec. 2020, pp. 458–467. DOI: 10.1109/TrustCom50675.2020.00069 (cit. on p. 5).

[Bau16] Christian Baun. "Mobile Clusters of Single Board Computers: An Option for Providing Resources to Student Projects and Researchers." in: *SpringerPlus* 5.1 (2016), p. 360 (cit. on p. 29).

[BCW07] Matteo Berioli, Nicolas Courville, and Markus Werner. "Emergency Communications over Satellite: the WISECOM Approach." in: *16Th IST Mobile and Wireless Communications Summit*. IEEE. 2007, pp. 1–5 (cit. on p. 82).

[Bel05] Fabrice Bellard. "QEMU, a Fast and Portable Dynamic Translator." in: *USENIX Annual Technical Conference, FREENIX Track*. vol. 41. 2005, p. 46 (cit. on p. 31).

[Ber+14] Carlos J Bernardos, Antonio De La Oliva, Pablo Serrano, Albert Banchs, Luis M Contreras, Hao Jin, and Juan Carlos Zúñiga. "An Architecture for Software Defined Wireless Networking." in: *IEEE Wireless Communications* 21.3 (2014), pp. 52–61 (cit. on p. 184).

[BFB22] Scott Burleigh, Kevin Fall, and Edward J. Birrane. *Bundle Protocol Version 7*. RFC 9171. RFC Editor, Oct. 2022. URL: https://www.rfc-editor.org/rfc/rfc9171.txt (cit. on pp. 127, 129, 140, 144).

[BGZ17] Moritz Beller, Georgios Gousios, and Andy Zaidman. "Oops, my Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub." in: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE. 2017, pp. 356–367 (cit. on p. 33).

[BHM19] Lars Baumgärtner, Jonas Höchst, and Tobias Meuser. "B-DTN7: Browser-based Disruption-tolerant Networking via Bundle Protocol 7." in: *2019 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM'19)*. Paris, France, Dec. 2019. DOI: 10.1109/ICT-DM47966.2019.9032944 (cit. on p. 6).

[Bis+18] Pratik K Biswas, Sharon J Mackey, Derya H Cansever, Mitesh P Patel, and Frank B Panettieri. "Context-Aware Smallworld Routing for Wireless Ad-Hoc Networks." in: *IEEE Transactions on Communications* 66.9 (Sept. 2018), pp. 3943–3958. DOI: 10.1109/TCOMM.2018.2811486 (cit. on p. 141).

[Bol+07] Chiara Boldrini, Marco Conti, Iacopo Iacopini, and Andrea Passarella. "HiBOp: a History Based Routing Protocol for Opportunistic Networks." in: *2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. June 2007, pp. 1–12. DOI: 10.1109/WOWMOM.2007.4351716 (cit. on p. 141).

[Bon16] Drew Bonasera. *PiShrink: Make Your Pi Images Smaller!* 2016. URL: https://github.com/Drewsif/PiShrink (visited on 11/05/2019) (cit. on p. 28).

[BSC18] Kyung Min Baek, Dong Yeong Seo, and Yun Won Chung. "An Improved Opportunistic Routing Protocol Based on Context Information of Mobile Nodes." in: *Applied Sciences* 8.8 (2018). ISSN: 2076-3417. DOI: 10.3390/app8081344 (cit. on p. 142).

[BSS10] Agathe Battestini, Vidya Setlur, and Timothy Sohn. "A Large Scale Study of Text Messaging Use." in: *12th Int. Conf. on Human Computer Interaction with Mobile Devices and Services*. ACM. 2010, pp. 229–238 (cit. on p. 87).

[Bur07] Scott Burleigh. *Interplanetary Overlay Network An Implementation of the DTN Bundle Protocol*. tech. rep. JPL, 2007 (cit. on p. 128).

[Bur19] Scott Burleigh. *Minimal TCP Convergence-Layer Protocol*. tech. rep. IETF, 2019 (cit. on p. 131).

[Bux+18] Rachel T Buxton, Patrick E Lendrum, Kevin R Crooks, and George Wittemyer. "Pairing Camera Traps and Acoustic Recorders to Monitor the Ecological Impact of Human Disturbance." in: *Global Ecology and Conservation* 16 (2018), e00493 (cit. on p. 56).

[BVR16] Martin Bor, John Vidler, and Utz Roedig. "LoRa for the Internet of Things." in: *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*. EWSN '16. Graz, Austria: Junction Publishing, 2016, pp. 361–366. ISBN: 9780994988607 (cit. on p. 156).

[BZS17] Paolo Bellavista, Alessandro Zanni, and Michele Solimando. "A Migration-enhanced Edge Computing Support for Mobile Devices in Hostile Environments." in: *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE. 2017, pp. 957–962 (cit. on p. 115).

[Cab+10] Roy Cabaniss, Sanjay Madria, George Rush, Abbey Trotta, and Srinivasa S Vulli. "Dynamic Social Grouping based Routing in a Mobile Ad-Hoc Network." in: *2010 International Conference on High Performance Computing*. Dec. 2010, pp. 1–8. DOI: 10.1109/HIPC.2010.5713165 (cit. on p. 141).

[Cac+13] Angela Sara Cacciapuoti, Francesco Calabrese, Marcello Caleffi, Giusy Di Lorenzo, and Luigi Paura. "Human-mobility Enabled Wireless Networks for Emergency Communications during Special Events." in: *Pervasive and Mobile Computing* 9.4 (2013), pp. 472–483 (cit. on p. 82).

[Cag+10] Francesca Cagnacci, Luigi Boitani, Roger A Powell, and Mark S Boyce. "Animal Ecology Meets GPS-based Radiotelemetry: A Perfect Storm of Opportunities and Challenges." in: *Philosophical Transactions of the Royal Society B: Biological Sciences* 365.1550 (2010), pp. 2157–2162. DOI: https://doi.org/10.1098/rstb.2010.0107 (cit. on p. 41).

[Cal+19] Gilles Callebaut, Guus Leenders, Chesney Buyle, Stijn Crul, and Liesbet Van der Perre. "LoRa Physical Layer Evaluation for Point-to-Point Links and Coverage Measurements in Diverse Environments." in: *arXiv preprint arXiv:1909.08300* (2019) (cit. on p. 156).

[Car+17] Anthony Caravaggi, Peter B Banks, A Cole Burton, Caroline MV Finlay, Peter M Haswell, Matt W Hayward, Marcus J Rowcliffe, and Mike D Wood. "A Review of Camera Trapping for Conservation Behaviour Research." in: *Remote Sensing in Ecology and Conservation* 3.3 (2017), pp. 109–122 (cit. on p. 56).

[CDD11] Thomas Clausen, Christopher Dearlove, and Justin Dean. *RFC 6130: Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP)*. tech. rep. IETF, 2011 (cit. on p. 199).

[Cer+07] Vinton G. Cerf, Scott C. Burleigh, Robert C. Durst, Kevin Fall, Adrian J. Hooke, Keith L. Scott, Leigh Torgerson, and Howard S. Weiss. *Delay-Tolerant Networking Architecture*. tech. rep. RFC 4838. IETF, 2007 (cit. on p. 129).

[CH06] Louise Comfort and Thomas Haase. "Communication, Coherence, and Collective Action: The Impact of Hurricane Katrina on Communications Infrastructure." in: *Public Works Management & Policy* 10.4 (2006), pp. 328–343 (cit. on p. 81).

[Cha+16] Dimitris Chatzopoulos, Mahdieh Ahmadi, Sokol Kosta, and Pan Hui. "Have You Asked Your Neighbors? A Hidden Market Approach for Device-to-device Offloading." in: *17th IEEE Int. Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE. 2016, pp. 1–9 (cit. on p. 115).

[Che+13] Yung-Chih Chen, Yeon-sup Lim, Richard J. Gibbens, Erich M. Nahum, Ramin Khalili, and Don Towsley. "A Measurement-based Study of MultiPath TCP Performance over Wireless Networks." in: *Internet Measurement Conference*. ACM, 2013. DOI: 10.1145/2504730.2504751 (cit. on p. 214).

[Che+15] Jordan Cheney, Ben Klein, Anil K Jain, and Brendan F Klare. "Unconstrained Face Detection: State of the Art Baseline and Challenges." in: *International Conference on Biometrics (ICB '15)*. IEEE. 2015, pp. 229–236 (cit. on p. 107).

[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 '16. 2016, pp. 122–131 (cit. on pp. 96, 97).

[Che+16b] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. "Efficient Multi-user Computation Offloading for Mobile-edge Cloud Computing." in: *IEEE/ACM Transactions on Networking* 5 (2016), pp. 2795–2808 (cit. on p. 115).

[Cho+18] Mahfuzulhoq Chowdhury, Eckehard Steinbach, Wolfgang Kellerer, and Martin Maier. "Context-Aware Task Migration for HART-Centric Collaboration over WiFi Based Tactile Internet Infrastructures." in: *IEEE Transactions on Parallel and Distributed Systems* 29.6 (2018), pp. 1231–1246 (cit. on p. 115).

[Coh99] Jeffrey P Cohn. "Tracking Wildlife: High-tech Devices help Biologists Trace the Movements of Animals Through Sky and Sea." in: *BioScience* 49.1 (1999), pp. 12–17 (cit. on p. 41).

[Con+10] Marco Conti, Silvia Giordano, Martin May, and Andrea Passarella. "From Opportunistic Networks to Opportunistic Computing." in: *IEEE Communications Magazine* 48.9 (2010), pp. 126–139 (cit. on p. 113).

[CS10] Tim Clutton-Brock and Ben C Sheldon. "Individuals and Populations: The Role of Long-term, Individual-based Studies of Animals in Ecology and Evolutionary Biology." in: *Trends in ecology & evolution* 25.10 (2010), pp. 562–573. DOI: https://doi.org/10.1016/j.gecco.2018.e00493 (cit. on p. 55).

[CS14] Harsha Chenji and Radu Stoleru. "Delay-tolerant Networks (DTNs) for Emergency Communications." in: *Advances in Delay-tolerant Networks (DTNs): Architecture and Enhanced Performance* (2014), p. 105 (cit. on p. 82).

[Dai18] Kyle Daigle. *GitHub Actions: Built by You, Run by Us*. GitHub. Oct. 2018 (cit. on p. 33).

[Dar+18] Kevin Darras, Péter Batáry, Brett Furnas, Antonio Celis-Murillo, Steven L. Van Wilgenburg, Yeni A. Mulyani, and Teja Tscharntke. "Comparing the Sampling Performance of Sound Recorders Versus Point Counts in Bird Surveys: A Meta-Analysis." in: *Journal of Applied Ecology* 55.6 (2018), pp. 2575–2586. DOI: 10.1111/1365-2664.13229 (cit. on p. 65).

[DB17] Quentin De Coninck and Olivier Bonaventure. *Every Millisecond Counts: Tuning Multipath TCP for Interactive Applications on Smartphones*. tech. rep. Available at http://hdl.handle.net/2078.1/185717, 2017 (cit. on p. 214).

[DCC11] György Dán, Niklas Carlsson, and Ilias Chatzidrossos. "Efficient and Highly Available Peer Discovery: A Case for Independent Trackers and Gossipping." in: *2011 IEEE International Conference on Peer-to-Peer Computing (P2P)*. IEEE. 2011, pp. 290–299 (cit. on p. 198).

[De +16] Quentin De Coninck, Matthieu Baerts, Benjamin Hesmans, and Olivier Bonaventure. "A First Analysis of Multipath TCP on Smartphones." in: *17th Int. Passive and Active Measurements Conference*. vol. 17. Springer, 2016. DOI: 10.1007/978-3-319-30505-9_5 (cit. on p. 221).

[Dee+19] D. C. Deepak, Alexandros Ladas, Yusuf Abdulrahman Sambo, Haris Pervaiz, Christos Politis, and Muhammad Ali Imran. "An Overview of Post-disaster Emergency Communication Systems in the Future Networks." in: *IEEE Wireless Communications* 26.6 (2019), pp. 132–139. ISSN: 15580687. DOI: 10.1109/MWC.2019.1800467 (cit. on p. 156).

[Dem+03] Michael Demmer, Eric Brewer, Kevin Fall, Sushant Jain, Melissa Ho, and Rabin Patra. *Implementing Delay Tolerant Networking*. tech. rep. Intel Research Berkeley and University of California, Berkeley, 2003 (cit. on p. 128).

[Den+15] Shuiguang Deng, Longtao Huang, Javid Taheri, and Albert Y Zomaya. "Computation Offloading for Service Workflow in Mobile Cloud Computing." in: *IEEE Transactions on Parallel and Distributed Systems* 26.12 (2015), pp. 3317–3329 (cit. on p. 115).

[DF07] Michael Demmer and Kevin Fall. "DTLSR: Delay Tolerant Routing for Developing Regions." in: *Proceedings of the 2007 Workshop on Networked Systems for Developing Regions*. NSDR '07. Kyoto, Japan: ACM, 2007, 5:1–5:6. ISBN: 978-1-59593-787-2. DOI: 10.1145/1326571.1326579 (cit. on p. 148).

[Dis+21] Simone Disabato, Giuseppe Canonaco, Paul G Flikkema, Manuel Roveri, and Cesare Alippi. "Birdsong Detection at the Edge with Deep Learning." in: *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE. 2021, pp. 9–16 (cit. on p. 65).

[Doe+08] Michael Doering, Sven Lahde, Johannes Morgenroth, and Lars Wolf. "IBR-DTN: An Efficient Implementation for Embedded Systems." in: *Third ACM Workshop on Challenged Networks*. ACM. 2008, pp. 117–120 (cit. on pp. 127–129, 135).

[Dos+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." in: *9th Int. Conference on Learning Representations, ICLR 2021, Austria*. 2021 (cit. on p. 65).

[DPC12] Alberto Dainotti, Antonio Pescape, and Kimberly C Claffy. "Issues and Future Directions in Traffic Classification." in: *IEEE Network* 26.1 (2012) (cit. on p. 184).

[Dul16] Alexandre Dulaunoy. *Forban: A P2P Application for Link-local and Local Area Networks*. 2016. URL: `https://github.com/adulau/Forban` (cit. on p. 128).

[Eli+18] Olakunle Elijah, Tharek Abdul Rahman, Igbafe Orikumhi, Chee Yen Leow, and MHD Nour Hindia. "An Overview of Internet of Things (IoT) and Data Analytics in Agriculture: Benefits and Challenges." in: *IEEE Internet of Things Journal* 5.5 (2018), pp. 3758–3773 (cit. on p. 154).

[Ena+19] Hiroto Enari, Haruka S Enari, Kei Okuda, Tetsuya Maruyama, and Kana N Okuda. "An Evaluation of the Efficiency of Passive Acoustic Monitoring in Detecting Deer and Primates in Comparison with Camera Traps." in: *Ecological Indicators* 98 (2019), pp. 753–762 (cit. on p. 56).

[Er-+17] Mohamed Er-rouidi, Houda Moudni, Hassan Faouzi, Hicham Mouncif, and Abdelkrim Merbouha. "A Fuzzy-Based Routing Strategy to Improve Route Stability in MANET Based on AODV." in: *Networked Systems*. Cham: Springer International Publishing, 2017, pp. 40–48. ISBN: 978-3-319-59647-1. DOI: `10.1007/978-3-319-59647-1_4` (cit. on p. 141).

[Erm+07] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. "Semi-Supervised Network Traffic Classification." in: *ACM SIGMETRICS Performance Evaluation Review*. vol. 35. 1. ACM. 2007, pp. 369–370 (cit. on pp. 185, 186).

[ES18] Hanan H Elazhary and Sahar F Sabbeh. "The W 5 Framework for Computation Offloading in the Internet of Things." in: *IEEE Access* 6 (2018), pp. 23883–23895 (cit. on p. 115).

[Fai08] Florian Fainelli. "The OpenWRT Embedded Development Framework." in: *Proceedings of the Free and Open Source Software Developers European Meeting*. 2008, p. 106 (cit. on p. 28).

[Fan+18] Wenhao Fan, Yuan'an Liu, Bihua Tang, Fan Wu, and Zhongbao Wang. "Computation Offloading based on Cooperations of Mobile Edge Computing-enabled Base Stations." in: *IEEE Access* 6 (2018), pp. 22622–22633 (cit. on p. 115).

[Fen+18] Jie Feng, Liqiang Zhao, Jianbo Du, Xiaoli Chu, and F Richard Yu. "Computation Offloading and Resource Allocation in D2D-Enabled Mobile Edge Computing." in: *2018 IEEE Int. Conf. on Communications (ICC)*. IEEE. 2018, pp. 1–6 (cit. on p. 115).

[Fer16] David Ferguson. *PiBakery: Easily Customise Raspbian*. 2016. URL: `https://www.pibakery.org/index.html` (visited on 11/05/2019) (cit. on p. 28).

[FHT10] Markus Fiedler, Tobias Hossfeld, and Phuoc Tran-Gia. "A Generic Quantitative Relationship between Quality of Experience and Quality of Service." in: *IEEE Network* 24.2 (2010), pp. 36–41 (cit. on p. 13).

[FLR16] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. "Computing with Nearby Mobile Devices: A Work Sharing Algorithm For Mobile Edge-clouds." in: *IEEE Transactions on Cloud Computing* 1 (2016), pp. 1–1 (cit. on p. 114).

[For+13] Alan Ford, Costin Raiciu, Mark Handley, and Olivier Bonaventure. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 6824. Internet Engineering Task Force, 2013. DOI: 10.17487/RFC6824 (cit. on p. 214).

[Fou14] National Science Foundation. "Partnerships for Innovation: Building Innovation Capacity (PFI: BIC)." in: *Program Solicitation* (2014), NSF14–NSF610 (cit. on p. 10).

[Fri+19] Nicolas Friess, Jörg Bendix, Martin Brändle, Roland Brandl, Stephan Dahlke, Nina Farwig, Bernd Freisleben, Hajo Holzmann, Hanna Meyer, Thomas Müller, Lars Opgenoorth, Carina Peter, Petra Quillfeldt, Christoph Reudenbach, Bernhard Seeger, Ralf Steinmetz, and Thomas Nauss. "Introducing Nature 4.0: A Sensor Network for Environmental Monitoring in the Marburg Open Forest." in: *Biodiversity Information Science and Standards* 2 (2019) (cit. on pp. 12, 39).

[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." in: (2016) (cit. on pp. 2, 12).

[Frö+18] Alexander Frömmgen, Denny Stohr, Boris Koldehofe, and Amr Rizk. "Don't Repeat Yourself: Seamless Execution and Analysis of Extensive Network Experiments." in: *14th Int. Conf. on Emerging Networking Experiments and Technologies (CoNEXT'18)*. 2018 (cit. on pp. 135, 147).

[FS13] Huber Flores and Satish Srirama. "Adaptive Code Offloading for Mobile Cloud Applications: Exploiting Fuzzy Sets and Evidence-based Learning." in: *4th ACM Workshop on Mobile Cloud Computing and Services*. ACM. 2013, pp. 9–16 (cit. on p. 115).

[FTH16] Colin Funai, Cristiano Tapparello, and Wendi Heinzelman. "Mobile to Mobile Computational Offloading in Multi-hop Cooperative Networks." in: *IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2016, pp. 1–7 (cit. on p. 115).

[FW15] Marius Feldmann and Felix Walter. "µPCN - A Bundle Protocol Implementation for Microcontrollers." in: *2015 Int. Conf. on Wireless Communications & Signal Processing (WCSP)*. IEEE. 2015 (cit. on pp. 127, 128).

[Gal+21] Sarah Gallacher, Duncan Wilson, Alison Fairbrass, Daniyar Turmukhambetov, O Mac Aodha, Stefan Kreitmayer, M Firman, Gabriel Brostow, and Kate Jones. "Shazam for Bats: Internet of Things for Continuous Real-Time Biodiversity Monitoring." in: *IET Smart Cities* (2021) (cit. on p. 65).

[Gar+12] Paul Gardner-Stephen, Jeremy Lakeman, Romana Challans, Corey Wallis, Ariel Stulman, and Yoram Haddad. "MeshMS: Ad Hoc Data Transfer within Mesh Network." in: *International Journal of Communications, Network and System Sciences* 8.5 (2012), pp. 496–504 (cit. on pp. 81, 84, 104).

[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 (cit. on pp. 81, 84, 104).

[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: *IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE. 2013, pp. 162–166 (cit. on pp. 81, 83, 104).

[Gar11] Paul Gardner-Stephen. "The Serval Project: Practical Wireless Ad-Hoc Mobile Telecommunications." in: *Flinders University, Adelaide, South Australia, Tech. Rep* (2011) (cit. on pp. 81–83, 113, 118, 127, 128, 135, 154).

[GCG21] Yuan Gong, Yu-An Chung, and James R. Glass. "AST: Audio Spectrogram Transformer." in: *Interspeech 2021*. 2021, pp. 571–575. DOI: 10.21437/Interspeech.2021-698 (cit. on p. 65).

[Gor+12] Mark S Gordon, Davoud Anoushe Jamshidi, Scott A Mahlke, Zhuoqing Morley Mao, and Xu Chen. "COMET: Code Offload by Migrating Execution Transparently." in: *OSDI*. vol. 12. 2012, pp. 93–106 (cit. on p. 114).

[Got+19] Jannis Gottwald, Ralf Zeidler, Nicolas Friess, Marvin Ludwig, Christoph Reudenbach, and Thomas Nauss. "Introduction of an Automatic and Open-Source Radio-Tracking System for Small Animals." in: *Methods in Ecology and Evolution* 10.12 (2019), pp. 2163–2172 (cit. on pp. 40, 41, 43, 50, 52, 56).

[Got+21] Jannis Gottwald, Patrick Lampe, Jonas Höchst, Nicolas Friess, Julia Maier, Lea Leister, Betty Neumann, Tobias Richter, Bernd Freisleben, and Thomas Nauss. "BatRack: An Open-source Multi-sensor Device for Wildlife Research." in: *Methods in Ecology and Evolution* (July 2021). DOI: 10.1111/2041-210X.13672 (cit. on pp. viii, 5, 7, 55).

[Gra+18a] Pablo Graubner, Patrick Lampe, Jonas Höchst, Lars Baumgärtner, Mira Mezini, and Bernd Freisleben. "Opportunistic Named Functions in Disruption-tolerant Emergency Networks." in: *ACM International Conference on Computing Frontiers 2018 (ACM CF 2018)*. Ischia, Italy: ACM, May 2018. DOI: 10.1145/3203217.3203234 (cit. on pp. ix, 6, 97, 121, 160).

[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: *12th ACM Int. Conf. on Distributed and Event-based Systems*. ACM. 2018, pp. 112–123 (cit. on p. 227).

[Gra19] Pablo Graubner. "Energy-efficient Transitional Near-* Computing." Doctoral dissertation. University of Marburg, Germany, 2019 (cit. on p. 12).

[GY19] Stefan Greif and Yossi Yovel. "Using On-board Sound Recordings to Infer Behaviour of Free-moving Wild Animals." in: *Journal of Experimental Biology* 222.Suppl_1 (2019), jeb184689 (cit. on p. 61).

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." in: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. DOI: 10.1109/CVPR.2016.90. (cit. on p. 65).

[Hec20] Luke Hecht. *Methods for Studying Wild Animals' Causes of Death*. Wild Animal Initiative, https://www.wildanimalinitiative.org/blog/cause-of-death-2. Accessed: 2020-11-02. Nov. 2020 (cit. on p. 43).

[Hei+13] Kurtis Heimerl, Kashif Ali, Joshua Blumenstock, Brian Gawalt, and Eric Brewer. "Expanding Rural Cellular Networks with Virtual Coverage." in: *10th USENIX Symp. on Netw. Systems Design & Implementation*. 2013, pp. 283–296 (cit. on p. 83).

[HHO14] Andrea Hess, Esa Hyytiä, and Jörg Ott. "Efficient Neighbor Discovery in Mobile Opportunistic Networking using Mobility Awareness." in: *2014 Sixth International Conference on Communication Systems and Networks (COMSNETS)*. IEEE. 2014, pp. 1–8 (cit. on p. 198).

[Hil+19] Andrew P. Hill, Peter Prince, Jake L. Snaddon, C. Patrick Doncaster, and Alex Rogers. "AudioMoth: A Low-cost Acoustic Device for Monitoring Biodiversity and the Environment." in: *HardwareX* 6 (2019), e00073. ISSN: 2468-0672. DOI: 10.1016/j.ohx.2019.e00073 (cit. on p. 72).

[HJG07] Sana Horrich, Sana Ben Jamaa, and Philippe Godlewski. "Adaptive Vertical Mobility Decision in Heterogeneous Networks." in: *3rd Int. Conf. on Wireless and Mobile Communications*. Mar. 2007, pp. 44–44. DOI: 10.1109/ICWMC.2007.16 (cit. on p. 213).

[HM17] Lorin Hochstein and Rene Moser. *Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way*. O'Reilly Media, Inc., 2017 (cit. on p. 29).

[Höc+17] Jonas Höchst, Lars Baumgärtner, Matthias Hollick, and Bernd Freisleben. "Unsupervised Traffic Flow Classification Using a Neural Autoencoder." in: *42nd Annual IEEE Conference on Local Computer Networks (LCN 2017)*. Singapore, Oct. 2017. DOI: 10.1109/LCN.2017.57 (cit. on pp. ix, 6, 186).

[Höc+19] Jonas Höchst, Artur Sterz, Alexander Frömmgen, Denny Stohr, Ralf Steinmetz, and Bernd Freisleben. "Learning Wi-Fi Connection Loss Predictions for Seamless Vertical Handovers Using Multipath TCP." in: *2019 IEEE 44th Conference on Local Computer Networks (LCN 2019)*. **Best Paper Award**. Osnabrück, Germany, Oct. 2019. DOI: 10.1109/LCN44214.2019.8990753. URL: https://umr-ds.github.io/seamcon (cit. on pp. ix, 6, 7, 213).

[Höc+20a] Jonas Höchst, Lars Baumgärtner, Franz Kuntke, Alvar Penning, Artur Sterz, and Bernd Freisleben. "LoRa-based Device-to-Device Smartphone Communication for Crisis Scenarios." in: *17th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2020)*. Blacksburg, Virginia, USA, May 2020 (cit. on pp. ix, 5, 7, 49, 155).

[Höc+20b] Jonas Höchst, Alvar Penning, Patrick Lampe, and Bernd Freisleben. "PIMOD: A Tool for Configuring Single-Board Computer Operating System Images." in: *2020 IEEE Global Humanitarian Technology Conference (GHTC 2020)*. Seattle, USA, Oct. 2020, pp. 1–8. DOI: 10.1109/GHTC46280.2020.9342928 (cit. on pp. viii, 5, 7, 27, 44, 56, 69).

[Höc+21] Jonas Höchst, Jannis Gottwald, Patrick Lampe, Julian Zobel, Thomas Nauss, Ralf Steinmetz, and Bernd Freisleben. "tRackIT OS: Open-source Software for Reliable VHF Wildlife Tracking." in: *51. Jahrestagung der Gesellschaft für Informatik INFORMATIK 2021, Berlin, Germany*. LNI. GI, Sept. 2021. DOI: 10.18420/informatik2021-035 (cit. on pp. viii, 5, 7, 41).

[Höc+22a] Jonas Höchst, Lars Baumgärtner, Franz Kuntke, Alvar Penning, Artur Sterz, Markus Sommer, and Bernd Freisleben. "Mobile Device-to-Device Communication for Crisis Scenarios Using Low-cost LoRa Modems." in: *Disaster Management and Information Technology: Professional Response and Recovery Management in the Age of Disasters*. ed. by Hans Jochen Scholl, Eric E. Holdeman, and F. Kees Boersma. Springer Nature, 2022 (cit. on pp. ix, 4, 7, 155).

[Höc+22b] Jonas Höchst, Hicham Bellafkir, Patrick Lampe, Markus Vogelbacher, Markus Mühling, Daniel Schneider, Kim Lindner, Sascha Rösner, Dana G. Schabo, Nina Farwig, and Bernd Freisleben. "Bird@Edge: Bird Species Recognition at the Edge." in: *International Conference on Networked Systems (NETYS)*. Springer. May 2022. DOI: 10.1007/978-3-031-17436-0_6 (cit. on pp. viii, 4, 5, 7, 64).

[Hor10] Craig A Hornbuckle. "Fractional-N Synthesized Chirp Generator." in: *United States Patent US7791415B2, Semtech Corp (May 2007)* (2010) (cit. on p. 154).

[Hos+16] Jan Hosang, Rodrigo Benenson, Piotr Dollár, and Bernt Schiele. "What Makes for Effective Detection Proposals?" in: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.4 (2016), pp. 814–830 (cit. on p. 106).

[Hos12] Ben Hosmer. "Getting Started with Salt Stack - The Other Configuration Management System built with Python." in: *Linux journal* 2012.223 (2012), p. 3 (cit. on p. 29).

[Hot+15] Torsten Hothorn, Jörg Müller, Leonhard Held, Lisa Möst, and Atle Mysterud. "Temporal Patterns of Deer–Vehicle Collisions Consistent with Deer Activity Pattern and Density Increase but not General Accident Risk." in: *Accident Analysis & Prevention* 81 (2015), pp. 143–152 (cit. on p. 40).

[HPS21] Christof Henkel, Pascal Pfeiffer, and Philipp Singer. "Recognizing Bird Species in Diverse Soundscapes under Weak Supervision." in: *Working Notes of CLEF 2021 - Conference and Labs of the Evaluation Forum, Bucharest, Romania, September 21-24, 2021*. ed. by Guglielmo Faggioli, Nicola Ferro, Alexis Joly, Maria Maistro, and Florina Piroi. vol. 2936. CEUR Workshop Proceedings. CEUR-WS.org, 2021, pp. 1579–1586. URL: http://ceur-ws.org/Vol-2936/paper-134.pdf (cit. on p. 65).

[Hua+14] Peihao Huang, Yan Huang, Wei Wang, and Liang Wang. "Deep Embedding Network for Clustering." in: *22nd International Conference on Pattern Recognition (ICPR)*. IEEE. 2014, pp. 1532–1537 (cit. on p. 188).

[iNa] iNaturalist. *A community for naturalists*. URL: https://www.inaturalist.org/ (cit. on p. 72).

[Jac+01] Philippe Jacquet, Paul Muhlethaler, Thomas Clausen, Anis Laouiti, Amir Qayyum, and Laurent Viennot. "Optimized Link State Routing Protocol for Ad Hoc Networks." in: *IEEE Int. Conf. on Technology for the 21st Century*. 2001, pp. 62–68 (cit. on p. 84).

[Jac+09] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. "Networking Named Content." in: *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*. CoNEXT '09. ACM, 2009, pp. 1–12 (cit. on p. 96).

[JGA13] Rahul Johari, Neelima Gupta, and Sandhya Aneja. "CACBR: Context Aware Community Based Routing for Intermittently Connected Network." in: *Proceedings of the 10th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, &; Ubiquitous Networks*. PE-WASUN '13. Barcelona, Spain: ACM, 2013, pp. 137–140. ISBN: 978-1-4503-2360-4. DOI: 10.1145/2507248.2507272 (cit. on p. 141).

[JLP15] Lenord Melvix JSM, 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 '16. 2015 (cit. on p. 97).

[JNA08] David Johnson, Ntsibane Ntlatlapa, and Corinna Aichele. "A Simple Pragmatic Approach to Mesh Routing using BATMAN." in: *2nd IFIP Int. Symp. on Wireless Comm. and Information Technology in Developing Countries*. 2008 (cit. on p. 84).

[Joh+18] Steven J Johnston, Philip J Basford, Colin S Perkins, Herry Herry, Fung Po Tso, Dimitrios Pezaros, Robert D Mullins, Eiko Yoneki, Simon J Cox, and Jeremy Singer. "Commodity Single Board Computer Clusters and Their Applications." in: *Future Generation Computer Systems* 89 (2018), pp. 201–212 (cit. on p. 29).

[KA20] Todd E Katzner and Raphaël Arlettaz. "Evaluating Contributions of Recent Tracking-based Animal Movement Ecology to Conservation Management." in: *Frontiers in Ecology and Evolution* 7 (2020), p. 519 (cit. on p. 40).

[KAB10] Mustsfa Bani Khalaf, Ahmed Y Al-Dubai, and William Buchanan. "A New Adaptive Broadcasting Approach for Mobile Ad hoc Networks." in: *6th Conference on Wireless Advanced (WiAD)*. IEEE. 2010, pp. 1–6 (cit. on p. 198).

[Kah+20] Stefan Kahl, Mary Clapp, W. Alexander Hopping, Hervé Goëau, Hervé Glotin, Robert Planqué, Willem-Pier Vellinga, and Alexis Joly. "Overview of BirdCLEF 2020: Bird Sound Recognition in Complex Acoustic Environments." in: *Working Notes of CLEF 2020 - Conference and Labs of the Evaluation Forum, Thessaloniki, Greece, September 22-25, 2020*. ed. by Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névéol. vol. 2696. CEUR Workshop Proceedings. CEUR-WS.org, 2020. URL: http://ceur-ws.org/Vol-2696/paper%5C_262.pdf (cit. on p. 65).

[Kah+21a] Stefan Kahl, Tom Denton, Holger Klinck, Hervé Glotin, Hervé Goëau, Willem-Pier Vellinga, Robert Planqué, and Alexis Joly. "Overview of BirdCLEF 2021: Bird Call Identification in Soundscape Recordings." in: *Working Notes of CLEF 2021 - Conference and Labs of the Evaluation Forum, Bucharest, Romania, September 21-24, 2021*. ed. by Guglielmo Faggioli, Nicola Ferro, Alexis Joly, Maria Maistro, and Florina Piroi. vol. 2936. CEUR Workshop Proceedings. CEUR-WS.org, 2021, pp. 1437–1450. URL: http://ceur-ws.org/Vol-2936/paper-123.pdf (cit. on p. 65).

[Kah+21b] Stefan Kahl, Connor M. Wood, Maximilian Eibl, and Holger Klinck. "BirdNET: A Deep Learning Solution for Avian Diversity Monitoring." in: *Ecological Informatics* 61 (2021), p. 101236. ISSN: 1574-9541. DOI: 10.1016/j.ecoinf.2021.101236 (cit. on pp. 65, 72, 73).

[Kai12] Andrius Kairiukstis. *BuildRaspbianImage: Build (and Cross-compile) Your Own Image for Raspberry Pi*. 2012. URL: https://github.com/andrius/build-raspbian-image/ (visited on 11/05/2019) (cit. on p. 28).

[Kau+18] Marc André Kaufhold, Nicola Rupp, Christian Reuter, Christoph Amelunxen, and Massimo Cristaldi. "112.Social: Design and Evaluation of a Mobile Crisis App for Bidirectional Communication between Emergency Services and Citizens." in: *26th European Conference on Information Systems: Beyond Digitization - Facets of Socio-Technical Change, ECIS 2018* (2018) (cit. on p. 156).

[Kay+11] Roland Kays, Sameer Tilak, Margaret Crofoot, Tony Fountain, Daniel Obando, Alejandro Ortega, Franz Kuemmeth, Jamie Mandel, George Swenson, Thomas Lambert, et al. "Tracking Animal Location and Activity with an Automated Radio Telemetry System in a Tropical Rainforest." in: *The Computer Journal* 54.12 (2011), pp. 1931–1948 (cit. on pp. 41, 42, 56, 58).

[KB15] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." in: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. ed. by Yoshua Bengio and Yann LeCun. 2015. URL: https://arxiv.org/abs/1412.6980 (cit. on pp. 70, 189).

[Kel08] Marcella J Kelly. "Design, Evaluate, Refine: Camera Trap Studies for Elusive Species." in: *Animal Conservation* 11.3 (2008), pp. 182–184 (cit. on p. 55).

[Kha+17] Murad Khan, Awais Ahmad, Shehzad Khalid, Syed Hassan Ahmed, Sohail Jabbar, and Jamil Ahmad. "Fuzzy based Multi-criteria Vertical Handover Decision Modeling in Heterogeneous Wireless Networks." in: *Multimedia Tools and Applications* 76.23 (2017), pp. 24649–24674. ISSN: 1573-7721. DOI: 10.1007/s11042-016-4330-1 (cit. on p. 213).

[Kim+08] Hyunchul Kim, Kimberly C Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. "Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices." in: *Proceedings of the 2008 ACM CoNEXT Conference*. ACM. 2008, 11:1–11:12 (cit. on p. 186).

[Kim+15] Suhwuk Kim, Yuki Urata, Yuki Koizumi, and Toru Hasegawa. "Power-saving NDN-based Message Delivery based on Collaborative Communication in Disasters." in: *The 21st IEEE International Workshop on Local and Metropolitan Area Networks*. 2015 (cit. on p. 97).

[KR16] R Kumar and M Pallikonda Rajasekaran. "An IoT based Patient Monitoring System using Raspberry Pi." in: *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)*. IEEE. 2016, pp. 1–4 (cit. on p. 26).

[Kum+13] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. "A Survey of Computation Offloading for Mobile Systems." in: *Mobile Networks and Applications* 18.1 (2013), pp. 129–140 (cit. on p. 113).

[Kun82] Thomas H Kunz. "Roosting Ecology of Bats." in: *Ecology of bats*. Springer, 1982, pp. 1–55 (cit. on p. 55).

[KW16] David Kayisire and Jiuchang Wei. "ICT Adoption and Usage in Africa: Towards an Efficiency Assessment." in: *Information Technology for Development* 22.4 (2016), pp. 630–653 (cit. on p. 154).

[Lam+17] Patrick Lampe, Lars Baumgärtner, Ralf Steinmetz, and Bernd Freisleben. "SmartFace: Efficient Face Detection on Smartphones for Wireless On-demand Emergency Networks." in: *24th Int. Conference on Telecommunications (ICT)*. IEEE. 2017, pp. 1–7 (cit. on pp. 103, 108, 113, 120).

[Lam+22a] Patrick Lampe, Markus Sommer, Artur Sterz, Jonas Höchst, Christian Uhl, and Bernd Freisleben. "ForestEdge: Unobtrusive Mechanism Interception in Environmental Monitoring." in: *2022 IEEE 47th Conference on Local Computer Networks (LCN 2022)*. Edmonton, Canada, Sept. 2022. DOI: 10.1109/LCN53696.2022.9843426 (cit. on p. 4).

[Lam+22b] Patrick Lampe, Markus Sommer, Artur Sterz, Jonas Höchst, Christian Uhl, and Bernd Freisleben. "Unobtrusive Mechanism Interception: Teaching an Old Dog New Tricks." in: *2022 IEEE 47th Conference on Local Computer Networks (LCN 2022)*. Edmonton, Canada, Sept. 2022. DOI: 10.1109/LCN53696.2022.9843536 (cit. on p. 4).

[LDS04] Anders Lindgren, Avri Doria, and Olov Schelén. "Probabilistic Routing in Intermittently Connected Networks." in: *Service Assurance with Partial and Intermittent Resources*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 239–254. ISBN: 978-3-540-27767-5. DOI: 10.1145/961268.961272 (cit. on p. 148).

[Lee+19] Daniel Lees, Tom Schmidt, Craig DH Sherman, Grainne S Maguire, Peter Dann, Glenn Ehmke, and Michael A Weston. "An Assessment of Radio Telemetry for Monitoring Shorebird Chick Survival and Causes of Mortality." in: *Wildlife Research* 46.7 (2019), pp. 622–627 (cit. on p. 40).

[Li+13] Bingdong Li, Jeff Springer, George Bebis, and Mehmet Hadi Gunes. "A Survey of Network Flow Applications." in: *Journal of Network and Computer Applications* 36.2 (2013), pp. 567–581 (cit. on pp. 184, 185).

[Lie+17] Patrick Lieser, Flor Alvarez, Paul Gardner-Stephen, Matthias Hollick, and Doreen Boehnstedt. "Architecture for Responsive Emergency Communications Networks." in: *2017 IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE. 2017, pp. 1–9 (cit. on pp. 156, 160).

[Lig17] Roger A Light. "Mosquitto: Server and Client Implementation of the MQTT Protocol." in: *Journal of Open Source Software* 2.13 (2017), p. 265 (cit. on p. 44).

[Lin+11] Jó Ágila Bitsch Link, Christoph Wollgarten, Stefan Schupp, and Klaus Wehrle. "Perfect Difference Sets for Neighbor Discovery: Energy Efficient and Fair." in: *3rd Extreme Conference on Communication: The Amazon Expedition*. ACM. 2011, 5:1–5:6 (cit. on p. 198).

[Lin+17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. "Focal Loss for Dense Object Detection." in: *2017 IEEE International Conference on Computer Vision (ICCV)* (Oct. 2017) (cit. on p. 70).

[Liu+15] Yue Liu, David R Bild, David Adrian, Gulshan Singh, Robert P Dick, Dan S Wallach, and Z Morley Mao. "Performance and Energy Consumption Analysis of a Delay-Tolerant Network for Censorship-Resistant Communication." in: *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM. 2015, pp. 257–266 (cit. on pp. 83, 154, 198).

[Llo82] Stuart Lloyd. "Least Squares Quantization in PCM." in: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137 (cit. on p. 186).

[LN18] Ariel K Lenske and Joseph J Nocera. "Field Test of an Automated Radio-Telemetry System: Tracking Local Space use of Aerial Insectivores." in: *Journal of Field Ornithology* 89.2 (2018), pp. 173–187 (cit. on p. 42).

[Loo11] James Loope. *Managing Infrastructure with Puppet: Configuration Management at Scale*. O'Reilly Media, Inc., 2011 (cit. on p. 29).

[Lut+19] Manisha Luthra, Boris Koldehofe, Jonas Höchst, Patrick Lampe, Ali Haider Rizvi, and Bernd Freisleben. "INetCEP: In-Network Complex Event Processing for Information-Centric Networking." in: *15th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2019)*. Cambridge, UK, Sept. 2019. DOI: 10.1109/ANCS.2019.8901877 (cit. on p. 6).

[LW02] Andy Liaw and Matthew Wiener. "Classification and Regression by RandomForest." in: *R News* 2.3 (2002), pp. 18–22 (cit. on p. 218).

[LWL08] Tsungnan Lin, Chiapin Wang, and Po-Chiang Lin. "A Neural-Network-based Context-aware Handoff Algorithm for Multimedia Computing." in: *ACM Trans. Multimedia Comput. Commun. Appl.* 4.3 (Sept. 2008), 17:1–17:23. ISSN: 1551-6857. DOI: 10.1145/1386109.1386110 (cit. on p. 213).

[LYD09] Sunho Lim, Chansu Yu, and Chita R Das. "RandomCast: An Energy-Efficient Communication Scheme for Mobile Ad Hoc Networks." in: *IEEE Transactions on Mobile Computing* 8.8 (2009), pp. 1039–1051 (cit. on p. 198).

[Ma+04] Li Ma, Fei Yu, Victor CM Leung, and Tejinder Randhawa. "A New Method to Support UMTS/WLAN Vertical Handover Using SCTP." in: *IEEE Wireless Communications* 11.4 (2004), pp. 44–51 (cit. on p. 213).

[Man+14] Milos Manic, Dumidu Wijayasekara, Kasun Amarasinghe, Joel Hewlett, Kevin Handy, Christopher Becker, Bruce Patterson, and Ronald Peterson. "Next Generation Emergency Communication Systems via Software Defined Networks." in: *Third GENI Research and Educational Experiment Workshop*. IEEE. 2014, pp. 1–8 (cit. on p. 82).

[Mar+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/ (cit. on pp. 70, 192).

[May02] Viktor Mayer-Schönberger. "Emergency Communications: The Quest for Interoperability in the United States and Europe." in: *Paper 2002-7, John F. Kennedy School of Government, Harvard University* (2002) (cit. on p. 82).

[MB07] Balakrishan S Manoj and Alexandra Hubenko Baker. "Communication Challenges in Emergency Response." in: *Communications of the ACM* 50.3 (2007), pp. 51–53 (cit. on p. 154).

[McF+15] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and Music Signal Analysis in Python." in: *Proceedings of the 14th python in science conference*. vol. 8. 2015 (cit. on p. 70).

[MCV17] Davide Magrin, Marco Centenaro, and Lorenzo Vangelista. "Performance Evaluation of LoRa Networks in a Smart City Scenario." in: *2017 IEEE International Conference on communications (ICC)*. ieee. 2017, pp. 1–7 (cit. on p. 174).

[Mdh+17] Afef Mdhaffar, Tarak Chaari, Kaouthar Larbi, Mohamed Jmaiel, and Bernd Freisleben. "IoT-based Health Monitoring via LoRaWAN." in: *IEEE EUROCON 2017 -17th International Conference on Smart Technologies, Ohrid, Macedonia, July 6-8, 2017*. ed. by Ljupco Karadzinov, Goga Cvetkovski, and Pero Latkoski. IEEE, 2017, pp. 519–524. DOI: 10.1109/EUROCON.2017.8011165 (cit. on p. 49).

[Med15] Alexandra Medina-Borja. *Editorial Column—Smart Things as Service Providers: A Call for Convergence of Disciplines to Build a Research Agenda for the Service Systems of the Future*. 2015 (cit. on p. 10).

[MEO17] Asmae Ait Mansour, Nourddine Enneya, and Mohamed Ouadou. "A Seamless Handover Based MIH-Assisted PMIPV6 in Heterogeneous Network (LTE-WiFi)." in: *2nd Int. Conf. on Big Data, Cloud and Applications*. Tetouan, Morocco: ACM, 2017, 67:1–67:5. ISBN: 978-1-4503-4852-2. DOI: 10.1145/3090354.3090423 (cit. on p. 213).

[Mer14] Dirk Merkel. "Docker: Lightweight Linux Containers for Consistent Development and Deployment." in: *Linux Journal* 2014.239 (2014), p. 2 (cit. on p. 28).

[MHM05] Mirco Musolesi, Stephen Hailes, and Cecilia Mascolo. "Adaptive Routing for Intermittently Connected Mobile Ad Hoc Networks." in: *Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*. June 2005, pp. 183–189. DOI: 10.1109/WOWMOM.2005.17 (cit. on p. 141).

[Mil+20] Markus Milchram, Marcela Suarez-Rubio, Annika Schröder, and Alexander Bruckner. "Estimating Population Density of Insectivorous Bats based on Stationary Acoustic Detectors: A Case Study." in: *Ecology and evolution* 10.3 (2020), pp. 1135–1144 (cit. on p. 56).

[MM06] Cecilia Mascolo and Mirco Musolesi. "SCAR: Context-aware Adaptive Routing in Delay Tolerant Mobile Sensor Networks." in: *Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing*. IWCMC '06. Vancouver, British Columbia, Canada: ACM, 2006, pp. 533–538. ISBN: 1-59593-306-9. DOI: 10.1145/1143549.1143656 (cit. on p. 141).

[MM09] Mirco Musolesi and Cecilia Mascolo. "CAR: Context-Aware Adaptive Routing for Delay-Tolerant Mobile Networks." in: *IEEE Transactions on Mobile Computing* 8.2 (Feb. 2009), pp. 246–260. ISSN: 1536-1233. DOI: 10.1109/TMC.2008.107 (cit. on p. 141).

[Mon+10] Robert A Montgomery, Gary J Roloff, Jay M Ver Hoef, and Joshua J Millspaugh. "Can We Accurately Characterize Wildlife Resource Use when Telemetry Data are Imprecise?" in: *The Journal of Wildlife Management* 74.8 (2010), pp. 1917–1925 (cit. on p. 41).

[Mon+14] Edo Monticelli, Benno M Schubert, Mayutan Arumaithurai, Xiaoming Fu, and KK Ramakrishnan. "An Information Centric Approach for Communications in Disaster Situations." in: *IEEE 20th Int. Workshop on Local Metropolitan Area Networks*. 2014, pp. 1–6 (cit. on pp. 96, 97, 104).

[Mor] Darius Morawiec. "sklearn-porter." Transpile Trained Scikit-learn Estimators to C, Java, JavaScript and Others. URL: `https://github.com/nok/sklearn-porter` (cit. on p. 220).

[MP05] Andrew W Moore and Konstantina Papagiannaki. "Toward the Accurate Identification of Network Applications." in: *International Workshop on Passive and Active Network Measurement*. Springer. 2005, pp. 41–54 (cit. on p. 186).

[MPI20] Massimo Merenda, Carlo Porcaro, and Demetrio Iero. "Edge Machine Learning for AI-enabled IoT devices: A Review." in: *Sensors* 20.9 (2020), p. 2533 (cit. on p. 65).

[Mti+13] Abderrahmen Mtibaa, Afnan Fahim, Khaled A Harras, and Mostafa H Ammar. "Towards Resource Sharing in Mobile Device Clouds: Power Balancing Across Mobile Devices." in: *ACM SIGCOMM Computer Communication Review*. vol. 43. 4. ACM. 2013, pp. 51–56 (cit. on p. 114).

[Müh+20] Markus Mühling, Jakob Franz, Nikolaus Korfhage, and Bernd Freisleben. "Bird Species Recognition via Neural Architecture Search." in: *Working Notes of CLEF 2020 - Conference and Labs of the Evaluation Forum, Thessaloniki, Greece, September 22-25, 2020*. ed. by Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névéol. vol. 2696. CEUR Workshop Proceedings. CEUR-WS.org, 2020. URL: `http://ceur-ws.org/Vol-2696/paper%5C_188.pdf` (cit. on p. 65).

[MV13] Vijay Mahadevan and Nuno Vasconcelos. "Biologically Inspired Object Tracking Using Center-Surround Saliency Mechanisms." in: *IEEE Trans. on Pattern Analysis and Machine Intelligence* 35.3 (2013), pp. 541–554 (cit. on p. 106).

[MZ05] Andrew W Moore and Denis Zuev. "Internet Traffic Classification Using Bayesian Analysis Techniques." in: *ACM SIGMETRICS Performance Evaluation Review*. vol. 33. 1. ACM. 2005, pp. 50–60 (cit. on pp. 185, 186).

[NA08] Thuy TT Nguyen and Grenville Armitage. "A Survey of Techniques for Internet Traffic Classification using Machine Learning." in: *IEEE Communications Surveys & Tutorials* 10.4 (2008), pp. 56–76 (cit. on pp. 184, 186, 187).

[NAS15] Neeraj Namdev, Shikha Agrawal, and Sanjay Silkari. "Recent Advancement in Machine Learning based Internet Traffic Classification." in: *Procedia Computer Science* 60 (2015), pp. 784–791 (cit. on p. 186).

[Nat+07] Essam Natsheh, Adznan B Jantan, Sabira Khatun, and Subramaniam Shamala. "Adaptive Optimizing of Hello Messages in Wireless Ad-Hoc Networks." in: *Int. Arab J. Inf. Technol.* 4.3 (2007), pp. 191–200 (cit. on p. 198).

[NGA07] Nidal Nasser, Sghaier Guizani, and Eyhab Al-Masri. "Middleware Vertical Handoff Manager: A Neural Network-Based Solution." in: *2007 IEEE International Conference on Communications*. June 2007, pp. 5671–5676. DOI: 10.1109/ICC.2007.940 (cit. on p. 213).

[Ngu+17] The An Binh Nguyen, Pratyush Agnihotri, Christian Meurisch, Manisha Luthra, Rahul Dwarakanath, Jeremias Blendin, Doreen Böhnstedt, Michael Zink, and Ralf Steinmetz. "Efficient Crowd Sensing Task Distribution Through Context-aware NDN-based Geocast." in: *42nd IEEE Conference on Local Computer Networks (LCN'17)*. Singapore: IEEE, 2017, pp. 52–60 (cit. on p. 97).

[Ngu+21] Johnny Nguyen, Karl Kesper, Gunter Kräling, Christian Birk, Peter Mross, Nico Hofeditz, Jonas Höchst, Patrick Lampe, Alvar Penning, Bastian Leutenecker-Twelsiek, Carsten Schindler, Helwig Buchenauer, David Geisel, Caroline Sommer, Ronald Henning, Pascal Wallot, Thomas Wiesmann, Björn Beutel, Gunter Schneider, Enrique Castro-Camus, and Martin Koch. "Repurposing CPAP Machines as Stripped-down Ventilators." in: *Scientific Reports* 11.1 (June 2021), pp. 1–9. DOI: 10.1038/s41598-021-91673-7 (cit. on p. 5).

[NK13] Ladislav Naďo and Peter Kaňuch. "Dawn Swarming in Tree-Dwelling Bats — An Unexplored Behaviour." in: *Acta chiropterologica* 15.2 (2013), pp. 387–392 (cit. on p. 55).

[NN08] Anthony J. Nicholson and Brian D. Noble. "BreadCrumbs: Forecasting Mobile Connectivity." in: *14th ACM Int. Conf. on Mobile Computing and Networking*. MobiCom '08. ACM, 2008, pp. 46–57 (cit. on p. 213).

[NZP11] Hervé Ntareme, Marco Zennaro, and Björn Pehrson. "Delay Tolerant Network on Smartphones: Applications for Communication-challenged Areas." in: *Proc. of the 3rd Extreme Conf. on Communication*. ACM. 2011, pp. 14–21 (cit. on p. 83).

[OB18] Martin K Obrist and Ruedi Boesch. "BatScope Manages Acoustic Recordings, Analyses Calls, and Classifies Bat Species Automatically." in: *Canadian Journal of Zoology* 96.9 (2018), pp. 939–954 (cit. on p. 58).

[OLG10] Soon Y Oh, Davide Lau, and Mario Gerla. "Content Centric Networking in Tactical and Emergency MANETs." in: *2010 IFIP Wireless Days*. 2010, pp. 1–5 (cit. on p. 96).

[Olt+13] Alexandru-Corneliu Olteanu, George-Daniel Oprina, Nicolae Tapus, and Sven Zeisberg. "Enabling Mobile Devices for Home Automation using ZigBee." in: *2013 19th International Conference on Control Systems and Computer Science*. May 2013, pp. 189–195. DOI: 10.1109/CSCS.2013.63 (cit. on p. 157).

[Ope19] Open Garden. *Firechat*. 2019. URL: https://www.opengarden.com/firechat/ (cit. on pp. 127, 128).

[Paa+12] Christoph Paasch, Gregory Detal, Fabien Duchene, Costin Raiciu, and Olivier Bonaventure. "Exploring Mobile/WiFi Handover with Multipath TCP." in: *ACM SIGCOMM Workshop Cellnet*. 2012. DOI: 10.1145/2342468.2342476 (cit. on p. 214).

[Pau+16] Partha Sarathi Paul, Bishakh Chandra Ghosh, Kingshuk De, Sujoy Saha, Subrata Nandi, Subhanjan Saha, Indrajit Bhattacharya, and Sandip Chakraborty. "On Design and Implementation of a Scalable and Reliable Sync System for Delay Tolerant Challenged Networks." in: *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*. Jan. 2016, pp. 1–8. DOI: 10.1109/COMSNETS.2016.7439949 (cit. on p. 198).

[Pec+15] Tommaso Pecorella, Luca Simone Ronga, Francesco Chiti, Sara Jayousi, and Laurent Franck. "Emergency Satellite Communications: Research and Standardization Activities." in: *IEEE Communications Magazine* 53.5 (2015), pp. 170–177 (cit. on p. 82).

[Ped+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. "Scikit-Learn: Machine Learning in Python." in: *Journal of Machine Learning Research* 12.Oct (2011), pp. 2825–2830. URL: http://scikit-learn.org/ (cit. on p. 218).

[PEK11] Christopher Pluntke, Lars Eggert, and Niko Kiukkonen. "Saving Mobile Device Energy with MultiPath TCP." in: *6th International Workshop on MobiArch*. ACM, 2011, pp. 1–6. DOI: 10.1145/1999916.1999918 (cit. on p. 214).

[Pen+19] Alvar Penning, Lars Baumgärtner, Jonas Höchst, Artur Sterz, Mira Mezini, and Bernd Freisleben. "DTN7: An Open-Source Disruption-tolerant Networking Implementation of Bundle Protocol 7." in: *18th International Conference on Ad Hoc Networks and Wireless (ADHOC-NOW 2019)*. Esch-sur-Alzette, Luxemburg, Oct. 2019. DOI: 10.1007/978-3-030-31831-4_14 (cit. on pp. ix, 6, 7, 39, 128, 140, 144, 164).

[Pen15] Fei Peng. "A Novel Adaptive Mobility-Aware MAC Protocol in Wireless Sensor Networks." in: *Wireless Personal Communications* 81.2 (2015), pp. 489–501 (cit. on p. 198).

[Per10] Charles Ed Perkins. *IP Mobility Support for IPv4, Revised*. RFC 5944. Internet Engineering Task Force, 2010. DOI: 10.17487/RFC5944 (cit. on p. 213).

[Pet+17] Juha Petäjäjärvi, Konstantin Mikhaylov, Marko Pettissalo, Janne Janhunen, and Jari Iinatti. "Performance of a Low-power Wide-area Network based on LoRa Technology: Doppler Robustness, Scalability, and Coverage." in: *International Journal of Distributed Sensor Networks* 13.3 (2017), p. 1550147717699412 (cit. on p. 49).

[PK+15] P Devi Pradeep, B Anil Kumar, et al. "A Survey of Emergency Communication Network Architectures." in: *International Journal of u-and e-Service, Science and Technology* 8.4 (2015), pp. 61–68 (cit. on p. 82).

[Pos11] Stefan Poslad. *Ubiquitous Computing: Smart Devices, Environments and Interactions*. John Wiley & Sons, 2011 (cit. on pp. 9, 10).

[Pöt+11] Wolf-Bastian Pöttner, Johannes Morgenroth, Sebastian Schildt, and Lars Wolf. "Performance Comparison of DTN Bundle Protocol Implementations." in: *6th ACM Workshop on Challenged Networks*. ACM. 2011, pp. 61–64 (cit. on p. 129).

[Psa+14] Ioannis Psaras, Lorenzo Saino, Mayutan Arumaithurai, KK Ramakrishnan, and George Pavlou. "Name-based Replication Priorities in Disaster Cases." in: *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2014, pp. 434–439 (cit. on p. 97).

[Pug21] Jean-Francois Puget. "STFT Transformers for Bird Song Recognition." in: *Working Notes of CLEF 2021 - Conference and Labs of the Evaluation Forum, Bucharest, Romania, September 21-24, 2021*. vol. 2936. CEUR Workshop Proceedings. CEUR-WS.org, 2021. URL: http://ceur-ws.org/Vol-2936/paper-137.pdf (cit. on p. 65).

[QA18] Cyril Paolo Quitevis and Charleston Dale Ambatali. "Feasibility of an Amateur Radio Transmitter Implementation using Raspberry Pi for a Low Cost and Portable Emergency Communications Device." in: *2018 IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE. 2018, pp. 1–6 (cit. on p. 26).

[Qin+15] Tao Qin, Lei Wang, Zhaoli Liu, and Xiaohong Guan. "Robust Application Identification Methods for P2P and VoIP Traffic Classification in Backbone Networks." in: *Knowledge-Based Systems* 82 (2015), pp. 152–162 (cit. on p. 185).

[R F20] R Foundation for Statistical Computing. *R: A Language and Environment for Computing (Version 3.6.3)*. 2020 (cit. on p. 60).

[Rai+12] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, Mark Handley, et al. "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP." in: *NSDI*. 2012. URL: https://dl.acm.org/citation.cfm?id=2228338 (cit. on p. 214).

[Ram94] Chet Ramey. "What's GNU: Bash - The GNU Shell." in: *Linux Journal* 1994.4es (1994), p. 13 (cit. on p. 32).

[Ras14] Raspberry Pi Foundation. *Raspbian: A Free Operating System based on Debian Optimized for the Raspberry Pi Hardware*. 2014. URL: https://www.raspberrypi.org/downloads/raspbian/ (visited on 11/05/2019) (cit. on p. 28).

[Ras16] Raspberry Pi Foundation. *pi-gen: Tool Used to Create the raspberrypi.org Raspbian Images*. 2016. URL: https://github.com/RPi-Distro/pi-gen (visited on 11/05/2019) (cit. on p. 28).

[RGH20] Erika Rosas, Felipe Garay, and Nicolas Hidalgo. "Context-aware Self-adaptive Routing for Delay-tolerant Networks in Disaster Scenarios." in: *Ad Hoc Networks* 102 (2020), p. 102095. ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2020.102095 (cit. on p. 142).

[RH10] George F Riley and Thomas R Henderson. "The NS-3 Network Simulator." in: *Modeling and Tools for Network Simulation*. Springer, 2010, pp. 15–34 (cit. on pp. 135, 174).

[Rig18] RightMesh. *Terra: Lightweight and Extensible DTN Library*. 2018. URL: https://github.com/RightMesh/Terra (cit. on p. 128).

[Rip+20] Simon P Ripperger, Gerald G Carter, Rachel A Page, Niklas Duda, Alexander Koelpin, Robert Weigel, Markus Hartmann, Thorsten Nowak, Jörn Thielecke, Michael Schadhauser, et al. "Thinking Small: Next-Generation Sensor Networks Close the Size Gap in Vertebrate Biologging." in: *PLoS Biology* 18.4 (2020), e3000655 (cit. on pp. 42, 61).

[RK18] Christoph Randler and Nadine Kalb. "Distance and Size Matters: A Comparison of Six Wildlife Camera Traps and Their Usefulness for Wild Birds." in: *Ecology and Evolution* 8.14 (2018), pp. 7151–7163 (cit. on p. 56).

[Rom+20] Marcelo Romero, Wided Guédria, Hervé Panetto, and Béatrix Barafort. "Towards a Characterisation of Smart Systems: A Systematic Literature Review." in: *Computers in industry* 120 (2020), p. 103224 (cit. on pp. 9, 10).

[Row+08] J Marcus Rowcliffe, Juliet Field, Samuel T Turvey, and Chris Carbone. "Estimating Animal Density Using Camera Traps Without the Need for Individual Recognition." in: *Journal of Applied Ecology* 45.4 (2008), pp. 1228–1236 (cit. on p. 56).

[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: *IEEE 7th Ann. Ubiquitous Computing, Electronics Mobile Communication Conf. (UEMCON)*. 2016, pp. 1–3 (cit. on p. 97).

[RP18] Anuradha Ravi and Sateesh K Peddoju. "Mobile Computation Bursting: An Application Partitioning and Offloading Decision Engine." in: *19th Int. Conference on Distributed Computing and Networking*. 46. ACM. 2018 (cit. on p. 114).

[Rus+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "ImageNet Large Scale Visual Recognition Challenge." in: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252 (cit. on p. 70).

[Saw+09] Hall Sawyer, Matthew J Kauffman, Ryan M Nielson, and Jon S Horne. "Identifying and Prioritizing Ungulate Migration Routes for Landscape-level Conservation." in: *Ecological Applications* 19.8 (2009), pp. 2016–2025 (cit. on p. 40).

[SB07] Keith L. Scott and Scott Burleigh. *Bundle Protocol Specification*. tech. rep. RFC 5050. IETF, 2007 (cit. on p. 128).

[Sch+11a] Sebastian Schildt, Johannes Morgenroth, Wolf-Bastian Pöttner, and Lars Wolf. "IBR-DTN: A Lightweight, Modular and Highly Portable Bundle Protocol Implementation." in: *Electronic Communications of the EASST* 37 (2011) (cit. on p. 135).

[Sch+11b] Dennis Schwerdel, David Hock, Daniel Günther, Bernd Reuther, Paul Müller, and Phuoc Tran-Gia. "ToMaTo - A Network Experimentation Tool." in: *International Conference on Testbeds and Research Infrastructures*. Springer. 2011, pp. 1–10 (cit. on p. 135).

[Sch+19] Ulrike E Schlägel, Johannes Signer, Antje Herde, Sophie Eden, Florian Jeltsch, Jana A Eccard, and Melanie Dammhahn. "Estimating Interactions Between Individuals from Concurrent Animal Movements." in: *Methods in Ecology and Evolution* 10.8 (2019), pp. 1234–1245 (cit. on p. 61).

[Sci+18] Luca Sciullo, Frederico Fossemo, Angelo Trotta, and Marco Di Felice. "LOCATE: A LoRa-based mObile emergenCy mAnagement sysTEm." in: *2018 IEEE Global Communications Conference (GLOBECOM)*. Dec. 2018, pp. 1–7. DOI: 10.1109/GLOCOM.2018.8647177 (cit. on p. 157).

[Shi+12] Cong Shi, Vasileios Lakafosis, Mostafa H Ammar, and Ellen W Zegura. "Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices." in: *13th ACM Int. Symposium on Mobile Ad Hoc Networking and Computing*. ACM. 2012, pp. 145–154 (cit. on p. 114).

[Sif+14] Manolis Sifalakis, Basil Kohler, Christopher Scherb, and Christian Tschudin. "An Information Centric Network for Computing the Distribution of Computations." in: *Proceedings of the 1st ACM Conference on Information-Centric Networking*. ICN '14. 2014, pp. 137–146 (cit. on p. 98).

[Sip+19] Brian Sipos, Michael Demmer, Joerg Ott, and Simon Perreault. *Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4*. tech. rep. IETF, 2019 (cit. on p. 131).

[Som+22] Markus Sommer, Jonas Höchst, Artur Sterz, Alvar Penning, and Bernd Freisleben. "ProgDTN: Programmable Disruption-tolerant Networking." in: *International Conference on Networked Systems (NETYS)*. Springer. May 2022. DOI: 10.1007/978-3-031-17436-0_13 (cit. on pp. ix, 5, 7, 140).

[Son+13] Chunfeng Song, Feng Liu, Yongzhen Huang, Liang Wang, and Tieniu Tan. "Auto-encoder Based Data Clustering." in: *Iberoamerican Congress on Pattern Recognition*. Springer. 2013, pp. 117–124 (cit. on p. 188).

[Soy+12] Tolga Soyata, Rajani Muraleedharan, Colin Funai, Minseok Kwon, and Wendi Heinzelman. "Cloud-vision: Real-time Face Recognition Using a Mobile-Cloudlet-Cloud Acceleration Architecture." in: *IEEE Symposium on Computers and Communications (ISCC 2012)*. IEEE. 2012, pp. 59–66 (cit. on p. 97).

[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 '05. Philadelphia, Pennsylvania, USA: ACM, 2005, pp. 252–259. ISBN: 1-59593-026-4. DOI: 10.1145/1080139.1080143 (cit. on p. 148).

[Sri+13] Mukundhan Srinivasan, Antony Venus AJ, Arun Neol Victor, Madhuri Narayanan, Sree Rakshaa SP, Vineeth Vijayaraghavan, et al. "GreenEduComp: Low Cost Green Computing System for Education in Rural India: A Scheme for Sustainable Development Through Education." in: *2013 IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE. 2013, pp. 102–107 (cit. on p. 26).

[STD20] Luca Sciullo, Angelo Trotta, and Marco Di Felice. "Design and Performance Evaluation of a LoRa-based Mobile Emergency Management System (LOCATE)." in: *Ad Hoc Networks* 96 (2020), p. 101993. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2019.101993. URL: http://www.sciencedirect.com/science/article/pii/S1570870518309004 (cit. on p. 157).

[Ste+17] Artur Sterz, Lars Baumgärtner, Ragnar Mogk, Mira Mezini, and Bernd Freisleben. "DTN-RPC: Remote Procedure Calls for Disruption-tolerant Networking." in: *IFIP Networking Conference*. IEEE. 2017, pp. 1–9 (cit. on p. 115).

[Ste+19] Artur Sterz, Lars Baumgärtner, Jonas Höchst, Patrick Lampe, and Bernd Freisleben. "OPPLOAD: Offloading Computational Workflows in Opportunistic Networks." in: *2019 IEEE 44th Conference on Local Computer Networks (LCN 2019)*. Osnabrück, Germany, Oct. 2019. DOI: 10.1109/LCN44214.2019.8990775 (cit. on pp. ix, 6, 114).

[Sto+16] Denny Stohr, Alexander Frömmgen, Jan Fornoff, Michael Zink, Alejandro Buchmann, and Wolfgang Effelsberg. "QoE Analysis of DASH Cross-Layer Dependencies by Extensive Network Emulation." in: *2016 Workshop on QoE-based Analysis and Management of Data Communication Networks*. ACM. 2016, pp. 25–30. DOI: 2940136.2940141 (cit. on p. 223).

[Sto+19] Dan Stowell, Tereza Petrusková, Martin Šálek, and Pavel Linhart. "Automatic Acoustic Identification of Individuals in Multiple Species: Improving Identification across Recording Conditions." in: *Journal of the Royal Society Interface* 16.153 (2019), p. 20180940 (cit. on p. 56).

[Tah+16] Rafaa Tahar, Amine Dhraief, Abdelfettah Belghith, Hassan Mathkour, and Rafik Braham. "Autonomous and Adaptive Beaconing Strategy for Multi-interfaced Wireless Mobile Nodes." in: *Wireless Communications and Mobile Computing* 16.12 (2016), pp. 1625–1641 (cit. on p. 198).

[Tan+17] Marion Lara Tan, Raj Prasanna, Kristin Stock, Emma Hudson-Doyle, Graham Leonard, and David Johnston. "Mobile Applications in Crisis Informatics Literature: A Systematic Review." in: *International Journal of Disaster Risk Reduction* 24.November 2016 (2017), pp. 297–311. ISSN: 22124209. DOI: 10.1016/j.ijdrr.2017.06.009. URL: http://dx.doi.org/10.1016/j.ijdrr.2017.06.009 (cit. on p. 156).

[Tay+17] Philip Taylor, Tara Crewe, Stuart Mackenzie, Denis Lepage, Yves Aubry, Zoe Crysler, George Finney, Charles Francis, Christopher Guglielmo, Diana Hamilton, et al. "The Motus Wildlife Tracking System: A Collaborative Research Network to Enhance the Understanding of Wildlife Movement." in: *Avian Conservation and Ecology* 12.1 (2017) (cit. on pp. 42, 56).

[TL19] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." in: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6105–6114. DOI: 1905.11946 (cit. on p. 70).

[TMR19] Daniella Teixeira, Martine Maron, and Berndt J van Rensburg. "Bioacoustic Monitoring of Animal Vocal Behavior for Conservation." in: *Conservation Science and Practice* 1.8 (2019), e72 (cit. on p. 63).

[Tol+20] Sivan Toledo, David Shohami, Ingo Schiffner, Emmanuel Lourie, Yotam Orchan, Yoav Bartan, and Ran Nathan. "Cognitive Map-based Navigation in Wild Bats Revealed by a New High-throughput Tracking System." in: *Science* 369.6500 (2020), pp. 188–193 (cit. on p. 61).

[Tör+06] B Uğur Töreyin, Yiğithan Dedeoğlu, Uğur Güdükbay, and A Enis Cetin. "Computer Vision Based Method for Real-time Fire and Flame Detection." in: *Pattern Recognition Letters* 27.1 (2006), pp. 49–58 (cit. on p. 101).

[Tri+11] Sacha Trifunovic, Bernhard Distl, Dominik Schatzmann, and Franck Legendre. "WiFi-Opp: Ad-Hoc-less Opportunistic Networking." in: *6th ACM Workshop on Challenged Networks*. ACM. 2011, pp. 37–42 (cit. on p. 199).

[TRM12] Josh Thomas, Jeff Robble, and Nick Modly. "Off Grid Communications with Android Meshing the Mobile World." in: *2012 IEEE Conference on Technologies for Homeland Security*. 2012, pp. 401–405 (cit. on p. 82).

[Tro+15] Edgar Marko Trono, Yutaka Arakawa, Morihiko Tamai, and Keiichi Yasumoto. "DTN MapEx: Disaster Area Mapping through Distributed Computing over a Delay-tolerant Network." in: *2015 Eighth International Conference on Mobile Computing and Ubiquitous Networking (ICMU)*. IEEE. 2015, pp. 179–184 (cit. on p. 135).

[Tru+19] Seth Truitt, Tmothy D Gage, Benjamin E Vincent, and Seunghyun Chun. "Low-cost Remote Monitoring System for Small-Scale UPS Installations in Developing Countries." in: *2019 IEEE Global Humanitarian Technology Conference (GHTC)*. 2019, pp. 1–6 (cit. on p. 26).

[TS13] Christian Tschudin and Manolis Sifalakis. "Named Functions for Media Delivery Orchestration." in: *20th International Packet Video Workshop 2013*. 2013 (cit. on p. 98).

[TS14] Christian Tschudin and Manolis Sifalakis. "Named Functions and Cached Computations." in: *IEEE 11th Consumer Communications and Networking Conference (CCNC)*. 2014, pp. 851–857 (cit. on pp. 96–98).

[UQ22] Saif Ullah and Amir Qayyum. "Socially-aware Adaptive Delay-tolerant Network (DTN) Routing Protocol." in: *PLOS One* 17.1 (Jan. 2022), pp. 1–15. DOI: 10.1371/journal.pone.0262565 (cit. on p. 142).

[Vas+15] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. "Quality and Productivity Outcomes Relating to Continuous Integration in GitHub." in: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 2015, pp. 805–816 (cit. on p. 31).

[VB00] Amin Vahdat and David Becker. *Epidemic Routing for Partially Connected Ad Hoc Networks*. tech. rep. CS-2000-06. Duke University, 2000 (cit. on p. 148).

[VC74] Vladimir N Vapnik and Alexey J Chervonenkis. *Theory of Pattern Recognition*. Nauka, 1974 (cit. on p. 186).

[VCD16] Alina Vlăduţu, Dragoş Comăneci, and Ciprian Dobre. "Internet Traffic Classification based on Flows' Statistical Properties with Machine Learning." in: *International Journal of Network Management* (2016) (cit. on pp. 186, 189–192).

[Vin+10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion." in: *Journal of Machine Learning Research* 11.Dec (2010), pp. 3371–3408 (cit. on p. 232).

[VJ04] Paul Viola and Michael J Jones. “Robust Real-time Face Detection.” in: *International Journal of Computer Vision* 57.2 (2004), pp. 137–154 (cit. on p. 107).

[VT02] Maarten Van Steen and A Tanenbaum. “Distributed Systems Principles and Paradigms.” in: *Network* 2 (2002), p. 28 (cit. on p. 11).

[Wal+18] Zea Walton, Gustaf Samelius, Morten Odden, and Tomas Willebrand. “Long-distance Dispersal in Red Roxes Vulpes vulpes Revealed by GPS Tracking.” in: *European Journal of Wildlife Research* 64.6 (2018), pp. 1–6 (cit. on p. 41).

[Wan+07] Wei Wang, Weidong Gao, Xinyu Bai, Tao Peng, Gang Chuai, and Wenbo Wang. “A Framework of Wireless Emergency Communications based on Relaying and Cognitive Radio.” in: *IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications*. IEEE. 2007, pp. 1–5 (cit. on p. 82).

[Wan+11] Weetit Wanalertlak, Ben Lee, Chansu Yu, Myungchul Kim, Seung-Min Park, and Won-Tae Kim. “Behavior-based Mobility Prediction for Seamless Handoffs in Mobile Wireless Networks.” in: *Wireless Networks* 17.3 (2011), pp. 645–658. ISSN: 1022-0038. DOI: 10.1007/s11276-010-0303-x (cit. on p. 213).

[Wei+16] Adi Weller Weiser, Yotam Orchan, Ran Nathan, Motti Charter, Anthony J Weiss, and Sivan Toledo. “Characterizing the Accuracy of a Self-synchronized Reverse-GPS Wildlife Localization System.” in: *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE. 2016, pp. 1–12 (cit. on pp. 41, 42).

[Wei91] Mark Weiser. “The Computer for the 21st Century.” in: *Scientific American* 265.3 (1991), pp. 94–105 (cit. on p. 9).

[Wix+16] Andrew J Wixted, Peter Kinnaird, Hadi Larijani, Alan Tait, Ali Ahmadinia, and Niall Strachan. “Evaluation of LoRa and LoRaWAN for Wireless Sensor Networks.” in: *2016 IEEE SENSORS*. IEEE. 2016, pp. 1–3 (cit. on p. 156).

[WLJ14] Chuanmeizhi Wang, Yong Li, and Depeng Jin. “Mobility-assisted Opportunistic Computation Offloading.” in: *IEEE Communications Letters* 18.10 (2014), pp. 1779–1782 (cit. on p. 114).

[WO07] Darrell M. West and Marion Orr. “Race, Gender, and Communications in Natural Disasters.” in: *Policy Studies Journal* 35.4 (2007), pp. 569–586 (cit. on p. 81).

[WSM07] Wei Wang, Vikram Srinivasan, and Mehul Motani. “Adaptive Contact Probing Mechanisms for Delay Tolerant Applications.” in: *13th Annual ACM International Conference on Mobile Computing and Networking*. ACM. 2007, pp. 230–241 (cit. on p. 199).

[Wyc+18] Teal B Wyckoff, Hall Sawyer, Shannon E Albeke, Steven L Garman, and Matthew J Kauffman. “Evaluating the Influence of Energy and Residential Development on the Migratory Behavior of Mule Deer.” in: *Ecosphere* 9.2 (2018), e02113 (cit. on p. 41).

[Xen] Xeno-Canto. *Sharing Bird Sounds from Around the World*. URL: https://www.xeno-canto.org/ (cit. on p. 72).

[Xia+14] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. “A Survey on Software-defined Networking.” in: *IEEE Communications Surveys & Tutorials* 17.1 (2014), pp. 27–51 (cit. on p. 12).

[Yan+13] Lei Yang, Jiannong Cao, Yin Yuan, Tao Li, Andy Han, and Alvin Chan. "A Framework for Partitioning and Execution of Data Stream Applications in Mobile Cloud Computing." in: *ACM SIGMETRICS Performance Evaluation Review* 40.4 (2013), pp. 23–32 (cit. on p. 115).

[Yan+18] Lei Yang, Jiannong Cao, Zhenyu Wang, and Weigang Wu. "Network aware Mobile Edge Computation Partitioning in Multi-user Environments." in: *IEEE Transactions on Services Computing* (2018) (cit. on p. 115).

[YY17] Narasimha Saii Yamanoor and Srihari Yamanoor. "High Quality, Low Cost eEucation with the Raspberry Pi." in: *2017 IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE. 2017, pp. 1–5 (cit. on p. 26).

[Zan+17] Alessandro Zanni, Se-Young Yu, Paolo Bellavista, Rami Langar, and Stefano Secci. "Automated Selection of Offloadable Tasks for Mobile Computation Offloading in Edge Computing." in: *2017 13th international conference on network and service management (CNSM)*. IEEE. 2017, pp. 1–5 (cit. on p. 115).

[Zha+13] Jun Zhang, Yang Xiang, Wanlei Zhou, and Yu Wang. "Unsupervised Traffic Classification using Flow Statistical Properties and IP Packet Payload." in: *Journal of Comp. and Syst. Sciences* 79.5 (2013), pp. 573–585 (cit. on pp. 185–187, 189, 191, 192).

[Zha+15a] Bentao Zhang, Yong Li, Depeng Jin, Pan Hui, and Zhu Han. "Social-Aware Peer Discovery for D2D Communications Underlaying Cellular Networks." in: *IEEE Transactions on Wireless Communications* 14.5 (May 2015), pp. 2426–2439. ISSN: 1536-1276. DOI: 10.1109/TWC.2014.2386865 (cit. on p. 199).

[Zha+15b] Jun Zhang, Xiao Chen, Yang Xiang, Wanlei Zhou, and Jie Wu. "Robust Network Traffic Classification." in: *IEEE/ACM Transactions on Networking (TON)* 23.4 (2015), pp. 1257–1270 (cit. on p. 185).

[Zha+18] Daniel Zhang, Yue Ma, Yang Zhang, Suwen Lin, X Sharon Hu, and Dong Wang. "A Real-time and Non-cooperative Task Allocation Framework for Social Sensing Applications in Edge Computing Systems." in: *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2018, pp. 316–326 (cit. on p. 115).

[ZHS03] Rong Zheng, Jennifer C Hou, and Lui Sha. "Asynchronous Wakeup for Ad Hoc Networks." in: *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*. ACM. 2003, pp. 35–45 (cit. on p. 198).

[ZL17] Barret Zoph and Quoc V. Le. "Neural Architecture Search with Reinforcement Learning." in: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017 (cit. on p. 65).

[ZNW15] Yang Zhang, Dusit Niyato, and Ping Wang. "Offloading in Mobile Cloudlet Systems with Intermittent Connectivity." in: *IEEE Transactions on Mobile Computing* 14.12 (2015), pp. 2516–2529 (cit. on p. 114).

[Zob+21] Julian Zobel, Paul Frommelt, Patrick Lieser, Jonas Höchst, Patrick Lampe, Bernd Freisleben, and Ralf Steinmetz. "Energy-efficient Mobile Sensor Data Offloading via WiFi using LoRa-based Connectivity Estimations." in: *51. Jahrestagung der Gesellschaft für Informatik, INFORMATIK 2021, Berlin, Germany*. LNI. GI, Sept. 2021. DOI: 10.18420/informatik2021-037 (cit. on p. 5).

[Zua+21] Imran Zualkernan, Jacky Judas, Taslim Mahbub, Azadan Bhagwagar, and Priyanka Chand. "An AIoT System for Bat Species Classification." in: *2020 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*. 2021, pp. 155–160. DOI: 10.1109/IoTaIS50849.2021.9359704 (cit. on p. 66).

Curriculum Vitae

Angaben zur Person

Name	Jonas Höchst
Geburtsdatum, -ort	01.05.1992, Gießen

Ausbildung

2017 – 2022	Philipps-Universität Marburg Promotionsstudium Informatik Promotion zum Dr. rer. nat. Abschlussnote: 1,0 (Sehr gut)
2014 – 2017	Philipps-Universität Marburg Studiengang Informatik Abschluss mit akademischem Grad M.Sc. Abschlussnote: 13,2 Punkte
2011 – 2014	Philipps-Universität Marburg Studiengang Informatik Abschluss mit akademischem Grad B.Sc. Abschlussmote: 12,6 Punkte
2008 – 2011	Landgraf-Ludwigs-Gymnasium, Gießen Abschluss mit dem Zeugnis der Allgemeinen Hochschulreife

Berufserfahrung

2017 – 2022	Philipps-Universität Marburg Wissenschaftlicher Mitarbeiter in der AG Verteilte Systeme
2016	Philipps-Universität Marburg Studentische Hilfskraft für Forschung der AG Verteilte Systeme
2014 – 2016	Philipps-Universität Marburg Studentische Hilfskraft für Lehre
2013 – 2014	Philipps-Universität Marburg Studentische Hilfskraft für Forschung der AG Softwaretechnik
2012 – 2013	Justus-Liebig-Universität Gießen Studentische Hilfskraft für Forschung am Zentrum für Philosophie und Grundlagen der Wissenschaft