University of Kaiserslautern

Doctoral Thesis

Architectures and Methods for Large Scale Participatory Sensing and Data Modeling in Smart City Environments

Author:
Tobias Franke

Reviewers:
Prof. Dr. Paul Lukowicz
Prof. Dr. Albrecht Schmidt

Date of doctoral viva:
December 8, 2017

Dean:
Prof. Dr. Stefan Deßloch

Thesis approved by the Department of Computer Science of the University of Kaiserslautern (TU Kaiserslautern) for the award of the Doctoral Degree Doctor of Engineering (Dr.-Ing.)

Technische Universität Kaiserslautern

D 386

Declaration of Authorship

I, Tobias Franke, declare that this thesis titled, 'Architectures and Methods for Large Scale Participatory Sensing and Data Modeling in Smart City Environments' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____

Date: ____

"The brick walls are there for a reason. The brick walls are not there to keep us out. The brick walls are there to give us a chance to show how badly we want something. Because the brick walls are there to stop the people who don't want it badly enough. They're there to stop the other people."

RANDY PAUSCH, THE LAST LECTURE

UNIVERSITY OF KAISERSLAUTERN

Abstract

Department of Computer Science

Doctor of Engineering (Dr.-Ing.)

Architectures and Methods for Large Scale Participatory Sensing and Data Modeling in Smart City Environments

by Tobias Franke

The proliferation of sensors in everyday devices – especially in smartphones – has led to crowd sensing becoming an important technique in many urban applications ranging from noise pollution mapping or road condition monitoring to tracking the spreading of diseases. However, in order to establish integrated crowd sensing environments on a large scale, some open issues need to be tackled first. On a high level, this thesis concentrates on dealing with two of those key issues: (1) efficiently collecting and processing large amounts of sensor data from smartphones in a scalable manner and (2) extracting abstract data models from those collected data sets thereby enabling the development of complex smart city services based on the extracted knowledge.

Going more into detail, the first main contribution of this thesis is the development of methods and architectures to facilitate simple and efficient deployments, scalability and adaptability of crowd sensing applications in a broad range of scenarios while at the same time enabling the integration of incentivation mechanisms for the participating general public. During an evaluation within a complex, large-scale environment it is shown that real-world deployments of the proposed data recording architecture are in fact feasible. The second major contribution of this thesis is the development of a novel methodology for using the recorded data to extract abstract data models which are representing the inherent core characteristics of the source data correctly. Finally – and in order to bring together the results of the thesis – it is demonstrated how the proposed architecture and the modeling method can be used to implement a complex smart city service by employing a data driven development approach.

Acknowledgements

Firstly, I would like to thank Prof. Dr. Paul Lukowicz for having been such an inspiring PhD advisor. I admire him for always having a solution at hand for every conceivable problem. Without his insights and experience, none of this work would have been possible. I would especially like to thank him for believing in me and for having encouraged me to take steps that I probably wouldn't have taken otherwise. I am equally grateful for Prof. Dr. Albrecht Schmidt agreeing to co-examine my thesis – especially on such short notice.

I would also like to thank my colleagues, particularly Jens Weppner, Kamil Kloch, Gerald Pirkl, Agnes Grünerbl, Andreas Poxrucker, Peter Hevesi, Marco Hirsch, Jane Bensch and David Bannach. Working with them has always been great fun. Especially, I would like to thank Gernot Bahle for his friendship, endless counseling and proof-reading this thesis; and Gerald Bauer for proof-reading this thesis and most importantly for being a true friend even during times when I wasn't as available as I should have been.

From my time working at the SOCIONICAL research project I would like to wholeheartedly thank Prof. Eve Mitleton-Kelly from the London School of Economics, Rebecca Walker and Peter Clarke from the City of London Police, Dominic Reid, Pageant Master of the Lord Mayor's Show, and William Ross whose ingenuity and friendship I value very much. Without their continuous support, valuable insights and hands-on approach, this project and everything that followed afterwards wouldn't have been possible at all.

A large part of this thesis also wouldn't have been possible without the help of students with whom I enjoyed working together very much. I'd therefore like to thank Torben Schnuchel, Anton Dollmaier, Roman Bugaian, Olakunmi Joseph, Matthias von Treuberg and Josef Kinseher for the countless hours of great work they committed to their various projects.

I am grateful to Lazar Radenovic – to whom I owe a lot more than he is probably aware of – for proving that great friendships can persist regardless of geographical distance and of course for proof-reading this thesis. Equally, I would like to thank Peter Oehlmann for being a great friend and for always having been there during difficult times with truly meaningful advice.

I also wish to thank the investors of my company, Uwe Brand, Hansjörg Schäfer and Wolfgang Rück for actually having given me the freedom to finish my thesis and – most importantly – for believing in me.

My deepest gratitude goes to my family for their unconditional love and for always being there for me. I'd like to thank my grandfather for actually being the perfect grandfather and my father for having taught me so many valuable lessons in his own very special way and for always having my back regardless of how reckless some of my ideas may seem to him. And of course I would like to wholeheartedly thank my mother – I'm lacking the words to express my gratitude for everything she has done in order to ensure that I could lead a good life and become a decent person. For that I'm eternally grateful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Any fool can know. The point is to understand."

ALBERT EINSTEIN

The main purpose of this introductory chapter is to familiarize the reader with the topics discussed in this thesis and – most importantly – to put them into context. Furthermore, the contributions made to the research fields of crowd sensing and data modeling are presented. The last part of this chapter provides an overview about the structure of the thesis.

1.1 Motivation

Urbanization is widely regarded to be one of the main challenges of the 21st century. In the 2014 revision of its "World Urbanization Prospects" report [1] (latest version available), the Population Division of the United Nations Department of Economic and Social Affairs is stating that currently 54 % of the world's population is living in urban areas. In 1950, that number was as low as 30 % – by 2050 it is expected to rise up to 66 %. In total numbers, the urban population has grown from 746 million in 1950 to 3.9 billion in 2014. This trend is also confirmed by looking at the development of megacities, i.e. cities with 10 million or more inhabitants. In 2014 there was a total of 28 megacities – by 2030 this number will rise to 41.

Currently, Tokyo is the largest city in the world with about 38 million inhabitants, followed by Delhi with 25 million and Shanghai with 23 million. In 2030, Tokyo will keep that top spot but Delhi and Shanghai will have swiftly grown to a population of 36 million and 31 million inhabitants, respectively. This prognosis also sheds some light on one of the regions that will be most affected by urbanization over the next decades. While Asia and Africa are currently mostly rural with an urban population of 48 and 40 % respectively (North America: 82 %, South America: 80 %, Europe: 73 %), almost 90 % of the 2.5 billion people that are expected to be added to the world's urban population will stem from those two regions. In fact, the three countries of India, China and Nigeria alone are projected to account for 37 % of that growth. Figures 1.1 and 1.2 visualize those urbanization trends in more detail: Fig. 1.1 shows the projected growth rates of urban agglomerations on a global scale, while Fig. 1.2 compares the countries of Germany and China based on key urbanization metrics.

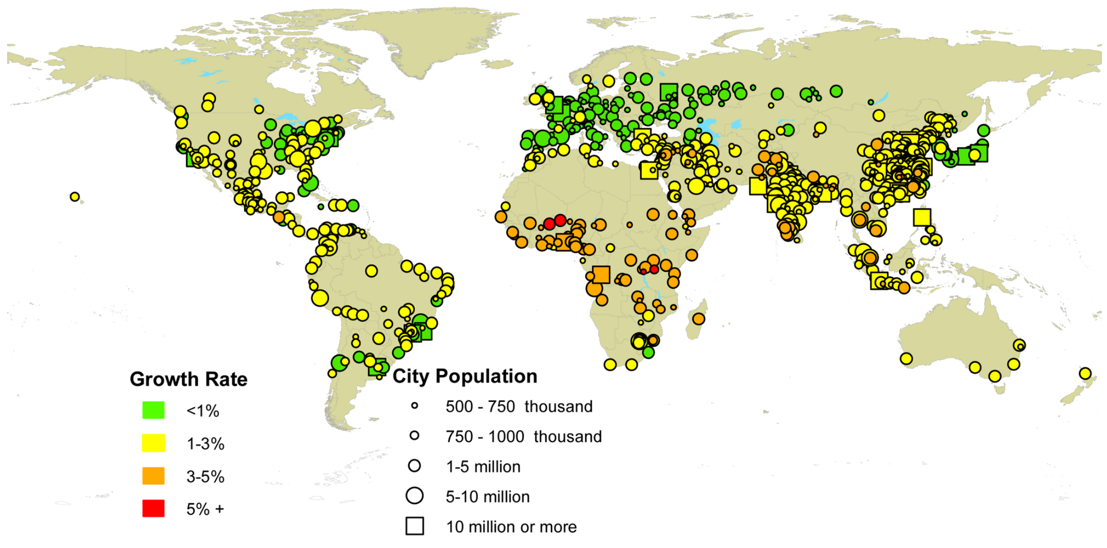


FIGURE 1.1: Projected growth rates of urban agglomerations by size class 2014 – 2030 *(©2014, United Nations, Population Division/DESA, www.unpopulation.org)*

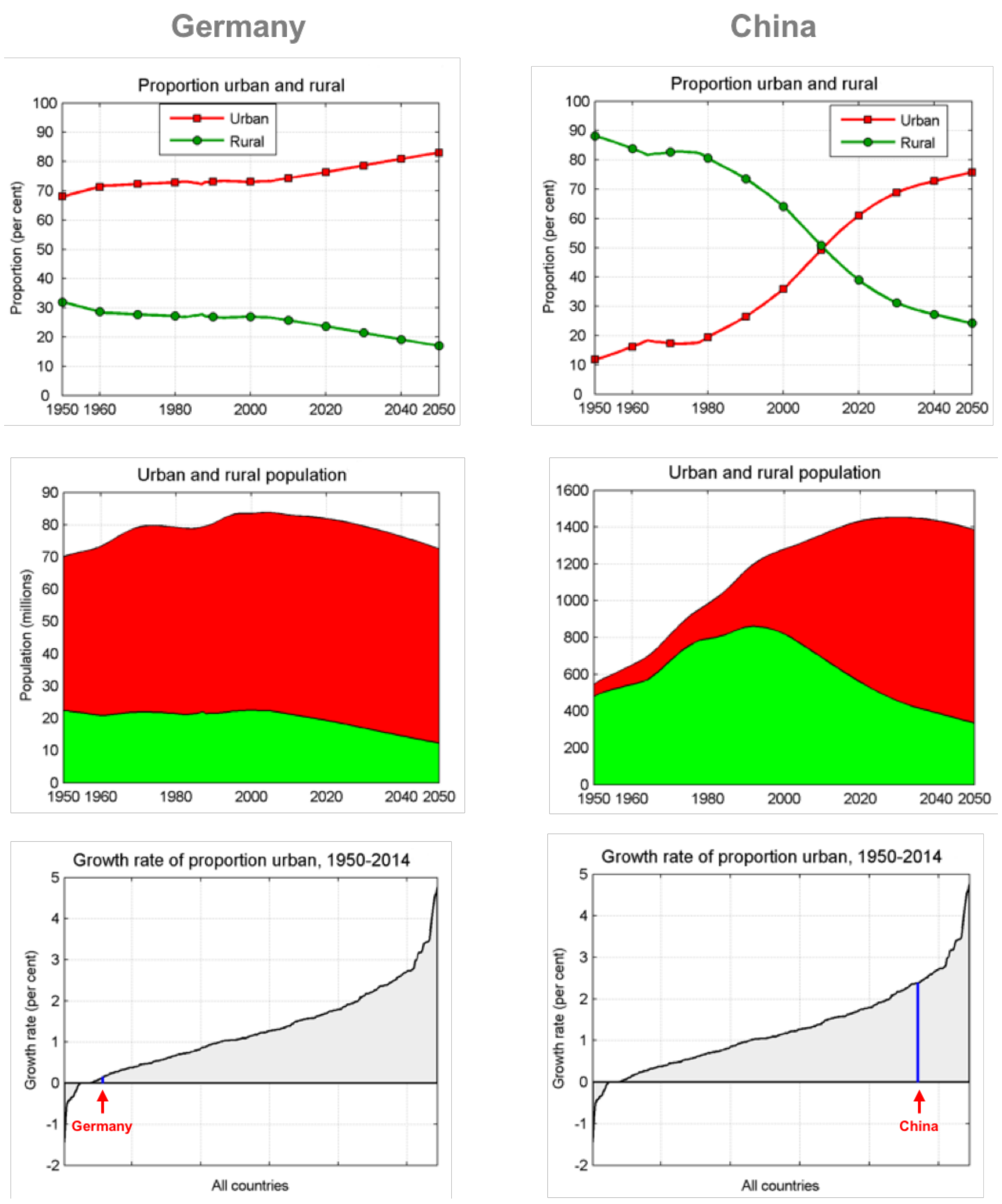


FIGURE 1.2: Urbanization metrics compared for Germany and China
(©2014, United Nations, Population Division/DESA, www.unpopulation.org)

On a positive note, urbanization generally holds many promises such as being able to make services like health care, public transportation, electricity, water, sanitation and education available to a larger population in an economically efficient manner. In other words, providing those services to urban households is both cheaper and more effective than doing the same in rural areas. The authors of [1] furthermore note that urban residents typically have access to a larger and more diversified job market.

However, there are also a number of challenges that come along with a fast-paced urbanization. Examples include overcrowding, destruction of the environment, pollution, potential health issues, safety and security threats, managing large amounts of traffic and increasing social inequality. For that reason, the United Nations report finishes with the words: *"Successful sustainable urbanization requires competent, responsive and accountable governments charged with the management of cities and urban expansion, as well as* ***appropriate use of information and communication technologies (ICTs) for more efficient service delivery.****"*

The highlighted part of the previous quote relates directly to the concept of Smart Cities – a paradigm of intelligent urban development, smart services and sustainable socio-economic growth [2]. Business analysts are expecting this field to be an important driver of the economy in the near future as they are estimating its global market value to reach a level of 1.5 trillion US$ by 2020 [3]. From an ICT point of view, a Smart City is leveraging data for gaining insights and for offering innovative services. [4] characterizes a Smart City as *"a system of systems"* – i.e. a collection of complex systems that need to be integrated efficiently. Examples for such sub-systems include transportation, energy and water, government services, public safety, education, healthcare and other ICT services. Consequently, a Smart City could for example react automatically to traffic congestions by dynamically re-routing cars and by adjusting traffic light switching strategies. Another application would be smart grids creating energy in a decentralized way to achieve higher efficiency. With respect to public health, a Smart City would constantly monitor parameters such as air quality and noise pollution in order to identify problematic areas as quickly as possible and react on that information immediately.

Within the research field of Smart Cities, one central question is how the data required for implementing all those services can be acquired and processed as efficiently as possible. Clearly, some areas do call for dedicated sensors to be installed in fixed locations. However, a large amount of applications can be fed with data recorded by means of crowd sensing – i.e. by citizens providing the data using hardware, they personally own. A very simple example would be a city-wide noise pollution map constructed from sound samples provided by the city's inhabitants. Smartphones in particular are ideal tools for that job, given their large array of built-in sensors [5, 6]. Also, the fact that they all have access to some kind of app store (and therefore to a distribution channel reaching hundreds of millions of users) helps enormously with making the required sensing software available to end users. Smartphones are especially suitable for crowd sensing systems as they are virtually ubiquitous nowadays. In Germany for example, about 60 % of the population owns such a device [7] – the most technologically savvy nations like South Korea achieve a smartphone penetration of about 73 %.

Previously, it was shown that a large part of the future urbanization is going to take place in China and India, which are generally regarded as low-income regions. However, despite of their comparatively low earnings, the majority of Chinese and Indian citizens also owns a smartphone (in China for example, the smartphone penetration is currently about 54 % [8]). In fact, it is quite likely, that those countries will catch up with the current leaders in the near future, as devices can be purchased there for less than 100 US$ [1]. According to the intelligence unit of the GSM Association, this leads to smartphones being the primary means for accessing the internet for the majority of people in the region [9] which can be considered as another big argument in favor of employing the concept of smartphone-based crowd sensing for realizing Smart City applications.

Crowd sensing has been an active field of research for the last ten years (see Chapter 1.2 for details) as it offers a whole host of advantages compared to traditional sensing approaches. For example, a very important upside is the fact that it doesn't require any additional infrastructure apart from a communication network – which is on its way to becoming a basic utility nowadays, anyway. Consequently, this approach doesn't call for large up-front investments in hardware and also doesn't require hardware-servicing. In addition to that, a crowd sensing system could theoretically be scaled up to the point where each citizen becomes a data contributor leading to a much better system coverage compared to a system based on fixed sensors (assuming sensible budgeting, of course).

Within this thesis, methods and architectures allowing for deployments of smartphone-based crowd sensing applications in a broad variety of scenarios are presented. While the data recorded with the resulting data collection platform could already be used for several analysis tasks in its raw form, this thesis aims for going one step further by presenting a novel methodology for extracting abstract models from that data. This is a meaningful step as those models present inherent "knowledge" contained within the raw data which can be a valuable asset for a Smart City. Therefore, the third contribution of this thesis is to demonstrate how such an extracted data model can be used to develop a novel Smart City service which significantly outperforms comparable approaches implemented in more traditional ways. Finally, in order to proof that the concepts elaborated in this work are actually feasible, they are evaluated within a complex, large-scale, real-world scenario.

[1] In China, the well-established brand Lenovo offers the reasonably well-equipped "Lenovo K3" for the small price of 95 US$ *(source:* `http://www.phonearena.com/news/Best-Chinese-Android-smartphones-May-2016_id63180`*)* – devices from lesser known brands can be had for even less.

1.2 Related Work

For the sake of not distracting the reader's focus from the respective topic being currently discussed, this thesis is structured in a way that each chapter contains its own "Related Work" section putting its corresponding contents and contributions into context. Consequently, this introductory related work section deals exclusively with the subject of crowd sensing in general in order to provide an overview about possible applications.

According to [5], crowd sensing can generally be applied at three different scales: (1) individual sensing, (2) group sensing and (3) community sensing. Individual sensing basically only records and evaluates data with respect to a single person — while that data may be put into relation to other users for the sake of comparisons (e.g. [10] where the users can measure their personal environmental impact and compare it to their friends' scores), it is not necessary to have a large number of participants in order for such systems to work. Group sensing on the other hand relies on data provided by a medium sized amount of users – e.g. for applications that focus on a group of friends like [11]. This thesis however, is putting the scale of community sensing at its center, where applications are deployed in large-scale environments like entire cities.

[5] also differentiates between participatory and opportunistic sensing. The former is defined to require the user's involvement in the sensing process, while the latter describes applications that are recording data without any user involvement whatsoever – i.e. the application is working transparently in the background. A perfect example for participatory sensing is given in [12] where the authors present a system that's processing text messages sent by residents of a city e.g. by means of a special app or via social media with the goal of identifying relevant events. The messages are processed in several stages using classic text processing techniques before those results are clustered with the goal of identifying related messages. The aggregated contents of these clusters are then for example presented to emergency personnel as additional source of information. During a 2-day trial-deployment in New York City, the system managed to identify 81 different events (ranging from accidents to a bank robbery) purely based on analyzing Twitter messages.

An example for opportunistic sensing is presented in [13]. Here, a system for monitoring road conditions is showcased. The authors implemented a smartphone app recording acceleration data once it recognizes that the device is inside a car. A classifier was trained to spot those sections of the signal where the car drives through a pot hole. The application then sends the location of that pothole to a central server where the data from all participating users is collected. Clusters of reported pot holes are computed and rated according to a scoring system. The locations of clusters that qualify as “real” pot holes are then published via a web service both for city planners and for drivers.

The topic of traffic is generally a very relevant one in the crowd sensing domain. In [14] an application is presented that uses a combination of crowd sensing and social media analysis to identify and characterize traffic anomalies. For spotting the locations of those anomalies, crowd sensed location data is analyzed. In contrast to similar works (e.g. [15]) however, the authors are not using the users’ velocities or the total traffic volume as features. Instead, they define anomalies to be areas where people’s routing differs significantly from the usual patterns. Once the anomaly areas are identified, social media posts published in the vicinity of that area are analyzed in order to determine the nature of the anomaly. The operators of the system are then presented with a map where the locations of all discovered anomalies are highlighted and associated with a tag cloud giving further information about the incident.

Another field of participatory crowd sensing applications is the topic of noise pollution mapping. [16] for example presents a smartphone application that allows its users to record a sound sample of their environment and upload it to a central server where a city-wide noise pollution map is created in order to identify dominant “noise hotspots”. In their work, the authors also point out one of the main challenges for all participatory sensing applications: the challenge of motivating users to actually participate.

[17] presents one possible solution for this challenge: so-called “gamification” where the task of contributing sensor data is turned into a game with the goal of users *wanting* to use the application because it’s fun. The scenario of [17] is also noise pollution mapping but in contrast to [16], the application is designed as a game with the result of a much larger user base. In order for gamification of a crowd sensing application to be successful, the authors define four key concepts which are partly interconnected:

- Status: users can collect points to increase their level.
- Access: the option to unlock new features (e.g. quests, new map areas, etc.).
- Power: “power users” can do more with the application than “novices”.
- Stuff: free rewards like badges or virtual gifts to keep motivation on a high level.

This thesis acknowledges the important topic of end user incentivation within the architecture of the proposed crowd sensing platform by allowing for multiple ways (incl. gamification) of motivating people to participate in data recording sessions.

Lately, the topic of medical research has also been introduced in the crowd sensing community. By collecting data points from a large number of patients in the real world, the resulting data set is much larger compared to clinical trials and the data is usually recorded under more realistic circumstances. [18] for example presents a crowd sensing application to reveal medical aspects about tinnitus. The authors argue that clinical trials are often cost- and labor-intensive. In addition to that, tinnitus is a condition that heavily depends on external and patient-internal factors which can't always be reproduced in a laboratory. The proposed application presents its users up to five times per day with questionnaires at random times. At the time of the writing of this thesis the study wasn't finished, but the authors already concluded that *"Altogether, using mobile crowd sensing and its application for psychological and medical trials offers promising perspectives."*. The importance of that topic is also underlined by the fact that industry heavy-weight Apple has presented its "ResearchKit" in 2015 [19] – a crowd sensing solution allowing medical clinics to perform large scale data collections. The first applications implemented with this tool were focusing on collecting data on asthma, cardiovascular disease, diabetes and Parkinson's disease.

Mitigating catastrophes by means of crowd sensing is also a very active research field. However, since this thesis is presenting an evaluation scenario from that domain, the respective related work will be presented in chapter 3.2.

With respect to scenario scale – and in fact vision – the upper end of crowd sensing would be an application working on a global level. [20] presents a research roadmap for exactly this scenario: a so-called "planetary nervous system (PNS)", which according to the authors should be *"capable of sensing and mining the digital breadcrumbs of human activities and unveiling the knowledge hidden in the big data for addressing the big questions about social complexity."*. The proposed system differs from previous works by aiming to include virtually all viable data sources ranging from stock brokering systems over smart sensors to social media for extracting knowledge on complex societal challenges. The authors *"envision the PNS as a globally distributed, self-organizing, techno-social system for answering analytical questions about the status of world-wide society"*.

1.3 Contributions

The main goal of this thesis is to develop methods, architectures and models to facilitate simple, efficient and broad deployments and utilizations of crowd sensing applications. The resulting concepts are evaluated within complex, large-scale, real-world environments. Thereby, the following research questions are answered:

1. What sort of architecture allows for quick and efficient deployments, scalability and adaptability of crowd sensing applications in a broad range of scenarios while at the same time enabling the integration of incentivation mechanisms for the participating public?
2. Are deployments of the proposed architecture feasible in the real world?
3. How can data models be extracted from the recorded data and are they representing the inherent core characteristics of the source data correctly?
4. How can complex smart services be developed based on those abstract data models?

With respect to research question (1), I elaborate in detail on the methods and design decisions made to ensure that the proposed crowd sensing platform has the following characteristics:

- Scalability – both from a technical and an operational point of view.
- Flexibility with respect to its deployment scenario.
- Expandability with respect to its feature set.
- The ability to be quickly deployable in large-scale scenarios – a typical deployment can be prepared within a day or two even if the scenario size includes tens of thousands of users, which is a key requirement of entities responsible for deploying such a platform.
- The ability to offer end users a concrete value for contributing data.
- The capability to reach as many end users as possible – the proposed platform has the potential to reach about 97 % of current smartphone users.

While a number of other crowd sensing platforms have been presented in the past (see chapter 2.2 for details), this work is the first to exhibit all of the characteristics mentioned above. It is therefore an approach that can actually be used in the real world.

One of the main challenges in assessing a crowd sensing platform's performance is its experimental evaluation under large-scale, real-life conditions. In this regard – and to answer research question (2) from above – I introduce the scenario of crowd management at large-scale events as a basis for an in-depth evaluation with respect to the platform's performance, user perception, influencing factors and usage. This evaluation is based on deployments with a total of more than 100'000 participants, which – to my best knowledge – is the largest user base achieved by any crowd sensing research platform to date.

Given a large enough number of participants, crowd sensing data recorded in a given scenario typically represents the observed phenomena within one specific instance of that scenario very well. Based on this observation and with respect to research question (3) from above, I furthermore present a methodology for extracting abstract data models from a recorded scenario instance. Those models can then be used to: (a) simulate other scenario instances with different configurations or (b) to evaluate new scenario-related applications. In general, such simulations and evaluations usually require a large number of "runs" to take the effect of statistical variations in the observed phenomena into account. However, in most scenarios it is not feasible to repeat and re-record data from a scenario instance many times over. Therefore, the proposed method allows for taking the data that was recorded once and using it to drive a simulation which replicates the core characteristics of the observed phenomena with appropriate statistical variations.

It is important to highlight the difference between the aims of this research and existing work on simulations. The majority of the latter use a global (statistical) model of the phenomena to be simulated. This thesis on the other hand shows how to extract different classes of model parameters based on a context-related separation of the underlying real-world data set's space. Therefore, this approach must be seen as complementary to classical simulations allowing them to initialize the model parameters for each sub-space of the scenario with realistic context-specific values.

For an evaluation of the approach, I revisit the scenario of crowd management at large-scale events: within this use case's context, an instance of the scenario corresponds to a certain event configuration with respect to time and space (e.g. location of concert stages or snack stands, times of performances, etc.) while the observed phenomenon is the crowd behavior. I demonstrate that by using the data recorded during one event day, it is possible to simulate the proceedings of another event day with a different configuration. Hence, it is not necessary to obtain another set of "training data" for that particular configuration. It is furthermore shown, that the general behavior of the simulated agents (i.e. general crowd conditions) corresponds to the real behavior expressed by people in the ground truth recording of the simulated event configuration, while at the same time realistic local behavior deviations (e.g. slightly different individual movement patterns while approaching an event location) are introduced, which leads to statistically relevant variations over the course of multiple simulation runs. In comparison to existing simulation methodologies, those deviations are not created by applying noise but by extracting realistic behavior patterns from the recorded data. Applying the proposed modeling methodology to the crowd management scenario leads to the following concrete contributions within the modeling and simulation domain:

- A spatio-temporal abstraction for dividing a city into appropriate context-related cells. The abstraction is based on an in-depth analysis of the recorded data set.
- Derivation of abstract crowd motion characteristics in each cell from the data set.
- A simulation system using such a characterization to generate actual crowd motion down to the level of individual agents.
- An evaluation of the methodology in terms of being able to simulate the crowd motion/distribution of each day of a large-scale event through a model derived from the other days.

Research question (4) from above is answered by presenting the exemplified development of a concrete smart city service in a data-driven fashion – i.e. by employing the methodologies elaborated in this thesis. As a show case for this, the development of an Ad-Hoc communication service was chosen. The novelty of this service is that its networking strategy is based on the mobility of its users. I show that by optimizing the parameters of the underlying algorithm using a large number of simulation runs created with the presented modeling approach, the communication service is working more efficiently than comparable services employing a traditional Ad-Hoc strategy – a fact which underlines the relevance of the "tool chain" presented in this thesis.

1.4 Thesis Overview

On a high level, this thesis consists of three parts (see Fig. 1.3): the first part deals with the topic of collecting data on a large scale and is divided into two chapters; one dealing with the technical side of the topic, the other evaluating the proposed architecture in a complex real-world scenario. The second main part of the thesis examines how knowledge can be extracted from the recorded data by means of creating abstract data models. In order to illustrate the application of the methods presented in this work, its third part demonstrates how a model extracted from a large scale crowd sensing data set can be used to develop a smart city service.

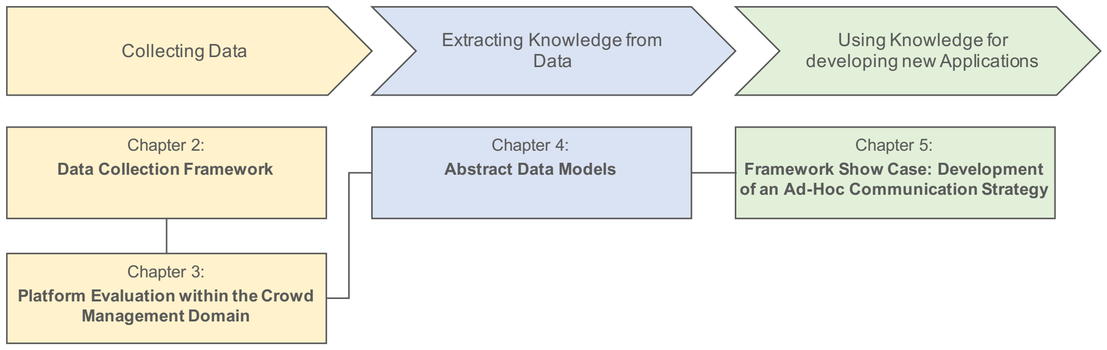

Collecting Data
Extracting Knowledge from Data
Using Knowledge for developing new Applications
Chapter 2: Data Collection Framework
Chapter 3: Platform Evaluation within the Crowd Management Domain
Chapter 4: Abstract Data Models
Chapter 5: Framework Show Case: Development of an Ad-Hoc Communication Strategy

FIGURE 1.3: Overview of thesis structure

Going into the specifics of this thesis' structure, **chapter 2** elaborates in detail on the architecture of the proposed crowd sensing platform. The individual parts are presented and design decisions are explained with respect to scalability and flexibility. After a general overview about the architecture's components is given, it is shown how a deployment is managed. Afterwards, the implementation details of relevant parts are presented. The chapter closes with analyzing the most important communication aspects between the main components.

chapter 3 continues the topic of large scale data collection by introducing the use case of crowd management as a complex real world scenario for evaluating the crowd sensing architecture presented in chapter 2. After the relevance and the specifics of the use case have been introduced, it is shown how the platform can be used to provide emergency personnel with situational awareness by means of crowd sensing and how it enables efficient crowd control by means of crowd steering via context specific message distribution. Afterwards, the feasibility of the proposed approach is verified. Furthermore, an overview about all deployments of the platform in the evaluation scenario is given. These deployments are then used as a basis for evaluating the crowd sensing platform with respect to insights into crowd behavior, user perception, influencing factors and usage.

chapter 4 presents a novel approach for extracting abstract data models from a crowd sensing data set. After the general methodology is explained, it is demonstrated in detail by applying it to the crowd management scenario. Afterwards, the quality of the resulting model is evaluated both qualitatively and quantitatively.

The main goal of **chapter 5** is to provide a real world showcase demonstrating how the concepts established in this thesis can be used to develop a novel smart city service in a data driven fashion. This is being exemplified by presenting the implementation of a mobility based Ad-Hoc communication system for smartphones. Before the actual concept of the system is discussed, the boundary conditions imposed by smartphones on such a system's performance are elaborated based on a series of experiments. Once the system's general behavior has been established, its performance-critical parameters are fine-tuned by using a pedestrian movement model created with the methodology from chapter 4 based on data recorded during one of the real-life deployments introduced in chapter 3. The chapter finishes by comparing the novel Ad-Hoc system to a more traditional approach.

chapter 6 provides a conclusion of the thesis by summarizing its contributions. Furthermore, this chapter is used to provide a critical look at the relevance and the limitations of the concepts covered in this work. Finally, an outlook on future research directions for the proposed methodologies is given.

The following Table 1.1 gives an overview about the peer-reviewed publications forming the basis of each thesis chapter:

Chapter	**Publications**
2	Franke, T., Lukowicz, P. and Blanke, U. (2015). **Smart crowds in smart cities: real life, city scale deployments of a smartphone based participatory crowd management platform.** *In Journal of Internet Services and Applications, 6(1), 1.*
3	Franke, T., Lukowicz, P., Wirz, M. and Mitleton-Kelly, E. (2013). **Participatory sensing and crowd management in public spaces.** *In Proceeding of the 11th annual international conference on Mobile systems, applications, and services (pp. 485-486). ACM.* AWARDED WITH THE BEST VIDEO AWARD Franke, T., Lukowicz, P. and Blanke, U. (2015). **Smart crowds in smart cities: real life, city scale deployments of a smartphone based participatory crowd management platform.** *In Journal of Internet Services and Applications, 6(1), 1.*

Chapter	Publications
	Wirz, M., Franke, T., Roggen, D., Mitleton-Kelly, E., Lukowicz, P. and Tröster, G. (2012). **Inferring crowd conditions from pedestrians' location traces for real-time crowd monitoring during city-scale mass gatherings.** *In Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on (pp. 367-372). IEEE.*
	Wirz, M., Franke, T., Roggen, D., Mitleton-Kelly, E., Lukowicz, P. and Tröster, G. (2013). **Probing crowd density through smartphones in city-scale mass gatherings.** *EPJ Data Science, 2(1), 1.*
	Blanke, U., Tröster, G., Franke, T. and Lukowicz, P. (2014). **Capturing crowd dynamics at large scale events using participatory gps-localization.** *In Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE 9th International Conference on (pp. 1-7). IEEE.*
4	Franke, T., Poxrucker, A., Bahle, G. and Lukowicz, P. (2016). **Trace Driven Simulation Model for City Scale Crowd Movements.** *In High Performance Computing and Communications; 14th International Conference on Smart City; 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on (pp. 1341-1348). IEEE.* AWARDED WITH THE BEST PAPER AWARD
5	Franke, T., Negele, S., Kampis, G. and Lukowicz, P. (2015). **Leveraging human mobility in smartphone based ad-hoc information distribution in crowd management scenarios.** *In 2015 2nd International Conference on Information and Communication Technologies for Disaster Management (ICT-DM) (pp. 27-34). IEEE.*
	Kampis, G., Franke, T., Negele, S. and Lukowicz, P. (2015). **Efficient Information Distribution Using Human Mobility.** *In Procedia Computer Science, 66, 382-391.*

Table 1.1: Publications building the basis for each thesis chapter.

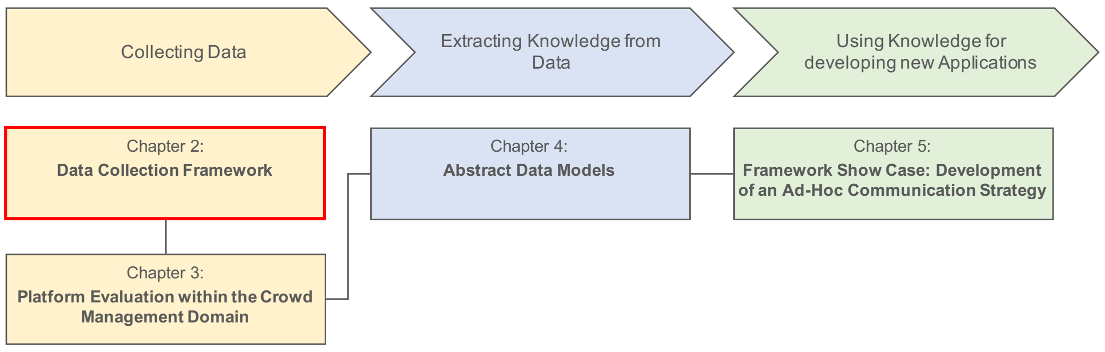
Chapter 2

Data Collection Platform

"The unexamined life is not worth living."

SOCRATES

This chapter builds the basis for the remainder of the thesis by presenting the architecture of the proposed crowd sensing platform. The individual components are presented and design decisions are explained with respect to influencing factors like scalability and flexibility. After a general overview about the platform's components and its deployment logic is given, the implementation details of the main parts are presented.


Collecting Data
Extracting Knowledge from Data
Using Knowledge for developing new Applications
Chapter 2: Data Collection Framework
Chapter 3: Platform Evaluation within the Crowd Management Domain
Chapter 4: Abstract Data Models
Chapter 5: Framework Show Case: Development of an Ad-Hoc Communication Strategy

2.1 Introduction

The basis of each crowd sensing application is a stable architecture for the recording and processing of data from a large number of participants. In the following chapter, I present an application-independent crowd sensing platform architecture which can be deployed in a wide range of real-world scenarios.

If such a platform is to be deployed successfully, it is of utmost importance to consider the requirements of the involved stakeholders: the participants required for collecting data and the entity responsible for the deployment. Henceforward, the latter will be referred to as "deployment owner". Participants are mainly concerned with getting an actual value in return for their willingness to share their data [16]. This value does not necessarily have to be monetary – it "just" has to be some form of motivation. Furthermore, they expect a crowd sensing system to protect their privacy [5]. Deployment owners on the other hand, are mostly interested in a quick and effortless deployment process, reasonable running costs and efficient tools to operate and manage the platform and the connected services. The architecture presented in this thesis covers all those requirements by combining data collection aspects and application-specific aspects in one common platform.

In essence, this thesis chapter answers the research question

> *What sort of architecture allows for quick and efficient deployments, scalability and adaptability of crowd sensing applications in a broad range of scenarios while at the same time enabling the integration of incentivation mechanisms for the participating public?*

by breaking it up into a series of smaller self-contained questions:

How can crowd sensing applications be efficiently deployed at scale?
It is shown how the architecture allows for quick roll-outs of the crowd sensing infrastructure – especially with respect to efficiently generating native, scenario-specific sensing applications for the participants.

How can a maximum degree of adaptability with respect to the type of deployment scenario be achieved?
One of the proposed architecture's main goals is to facilitate a broad range of crowd sensing applications. Therefore, the concepts pursued in order to reach this goal are presented in detail.

How can the application running on the participants' smartphones be designed in a way that it provides actual value to them?
As mentioned previously, motivating participants is vital for successful crowd sensing applications. The means to incorporate incentivation mechanisms into the system are presented – especially with respect to the architecture's capability to deliver application-specific contents to the participants.

How can scalability of all system parts be achieved in a cost-efficient manner in order to allow for feasible truly large-scale applications?
Since this thesis is concentrating on large-scale crowd sensing applications in real-world scenarios, it is important for the architecture to be able to deal with a growing number of users efficiently. The methods employed to reach that goal for each system part are therefore elaborated. An additional factor that has been considered in the scope of this question is the financial impact of a growing user base on the running costs.

The remainder of this chapter is structured as follows: the subsequent "Related Work" section is providing an overview about existing data collection frameworks and sets the work presented in this thesis in context. Next, an introduction into the high-level components of the proposed system and their interaction is given. The fourth section presents the deployment process of the system. The "System Architecture and Implementation" section elaborates the design decisions made for each component and presents the most important communication flows. The sixth and final section provides a summary of this chapter.

2.2 Related Work

The following section has the purpose of providing an overview about the work that has previously been performed in the field of crowd sensing with respect to data collection frameworks. Here, the focus is not on concrete crowd sensing applications (see chapter 1.2 for details on that topic) but rather on the infrastructure enabling such applications. Specifically, the focus is on data collection platforms that are independent from actual use cases and can therefore be deployed in multiple scenarios without major adaptations.

If one considers crowd sourcing as an early phase of crowd sensing, the case can be made for Amazon Mechanical Turk[1] being the first truly versatile data collection platform. It basically facilitates a marketplace where so-called Requesters can post tasks which are then solved by so-called Workers. Examples may include research groups requesting Workers to annotate a set of images in order to train a classifier. At the time of the writing

[1]`https://www.mturk.com`

of this thesis, Mechanical Turk had more than 773'000 members registered as Workers – a fact which underlines the potential of the platform for crowd sourcing applications. In 2009 (two years after the smartphone boom was triggered by the introduction of the original Apple iPhone) [21] proposed a smartphone application that utilized the Mechanical Turk platform for mobile crowd sourcing tasks. It allowed for applications such as asking citizens to take photos of problems in their neighborhoods. While this approach didn't allow for any kind of automatic processing of the collected data, it can in fact be seen as an early precursor of some of the Smart City crowd sensing applications introduced in chapter 1.2.

Generally speaking, most crowd sensing platforms follow a three-layer approach which has been elaborated in [5] and is made up of the following components:

1. The *Sense Layer* which is typically run on smartphones. This layer's main task is recording and possibly pre-processing data.
2. The *Learn Layer* running on cloud-based server infrastructure. This layer is responsible for processing the data contributed by all crowd sensing participants.
3. The *Inform Layer* which can take several forms such as social media channels or other web services. This layer's task is to present the results of the Learn Layer's analysis and is therefore the actual user interface of the crowd sensing application.

[5] also states that the concrete distribution of processing tasks between the Sense and the Learn Layer is quite variable. Three of the most important reasons for that are privacy concerns, battery drain and performance issues. With respect to the former, it is clear that a certain amount of on-device pre-processing of the raw data needs to take place before it can be sent to the cloud – at the very least some sort of anonymization needs to be performed. At the same time, the authors argue that too much processing on the phones is also no option as it might have too big an impact on the device's performance and battery drain. Finding the optimal balance for the separation of tasks between the Sense and Learn Layer is therefore still an ongoing research question.

In its overview about the development of mobile crowd sensing, [22] also presents a reference framework which is a good example for the basic three-layer-approach. [23] presents the MOSDEN platform which further develops the basic approach by introducing caching mechanisms to deal with situations where the smartphones might be offline. Furthermore, the framework allows for the integration of external sensors into the overall crowd sensing system. While MOSDEN does indeed present a reasonably general data collection platform, its large-scale deployment in the real world is hampered by (1) the fact that the sensing app is only available for Android devices; by (2) the lack of mechanisms for integrating incentives for the participants; and (3) by the fact that the framework does not have the ability to add application-specific components to the sensing app, which means that from the end users' perspective, the app itself is only a tool for sensing and does not provide any relevant information.

The authors of [24] are presenting the data recording platform MEDUSA. While the platform itself is not specifically geared towards the ability to be deployed in a wide range of real world scenarios, it does serve as a testbed for a new high level programming language and runtime system presented in that work. The goal of that language is to program crowd sensing tasks more efficiently. It furthermore allows incorporating human actions such as "take a picture" or "annotate the picture" at multiple stages into the workflow. MEDUSA also utilizes the Amazon Mechanical Turk platform for recruiting participants and for payments as incentivation mechanism. In general, developing a new programming language for crowd sensing tasks seems to be a valuable addition to the field as a whole but the underlying notion of the platform that sees crowd sensing as a series of tasks that need to be solved by the participants seems to go against the notion of everyday Smart City applications like they were presented in chapter 1.2.

However, that "task-based" understanding of crowd sensing is very much dominant when looking at existing application-independent data collection platforms. While many of them are indeed very flexible, they are mainly geared towards data collection in a research and development context rather than a real-world deployment context. [25] and [26] are two prime examples of such platforms. The defining feature of [25] is a geo-social model that's being used to profile the participants with the goal of asking only those users for participating in a task that are likely to accept it. While this concept is limited to learning the users' behavior with respect to time and space (i.e. estimating if that participant is likely to stay within the borders of the task's relevant area for the whole duration of the task) in the scope of the paper, it seems very feasible to extend it by adding further parameters like for instance user activity. Furthermore, it could also be applied to "non-task-based" applications, i.e. applications where data is continuously streamed without user intervention as it is the case in most Smart City scenarios.

The Participact Mobile Crowd Sensing Living Lab is introduced in [26]. The authors are presenting an Android-based research data collection framework with the notable ability to interface with external third party applications. To a certain degree it is therefore possible to use it in multiple scenarios by embedding it into scenario-related smartphone apps. The authors show the platform's capabilities by presenting a long-term deployment amongst 300 students. As a means of incentivation, the students earn points for contributing data. If a certain number of points has been earned, the students' data flatrate is being paid by the researchers. While the Participact platform does in fact have a very good scope with respect to the capability to be deployed in a range of scenarios, it does lack the means to quickly facilitate large-scale, real-world deployments and to create scenario-specific sensing apps (which should provide participants with actual value) efficiently.

Some related work has also been performed on more alternative approaches for individual parts of the generally accepted three-layer-approach introduced above. [27] for example proposes to use the Twitter infrastructure for the communication between the different components of a crowd sensing system. However, given the fact that Twitter has certain limits when it comes to the number of Tweets an individual user can post within a given time, it seems unlikely for this approach to scale well in large-scale scenarios. The authors of [28] also propose an alternative architecture by implementing a crowd sensing platform by employing a traditional publisher-subscriber pattern which is technically quite flexible but doesn't necessarily lead to a scalable large-scale system.

Compared with existing data collection platforms, the architecture developed within the course of this thesis has several novelties with respect to large-scale, real-world deployments:

- It allows for combining the core data collection aspects with application-specific aspects all within the same platform.
- It is able to quickly generate scenario-specific native sensing apps which provide actual value to the end users and which can be run by about 97 % of current smartphone users.
- It scales well both in technical terms (i.e. it's capable to deal with a large number of users) as well as in practical terms (i.e. it allows for an efficient management of large scenarios).
- It is geared towards the needs of both the participating end users as well as the entity responsible for deploying the application.
- It has specifically been designed for long-term deployments and therefore offers a range of tools to support daily operations and to provide end users with scenario-specific contents.

2.3 Architecture Overview

The following section introduces the proposed architecture's main components. As it's the case with most crowd sensing applications, the three-layer approach introduced in the previous "Related Work" section has been chosen as a basis. Figure 2.1 illustrates the system's main components.

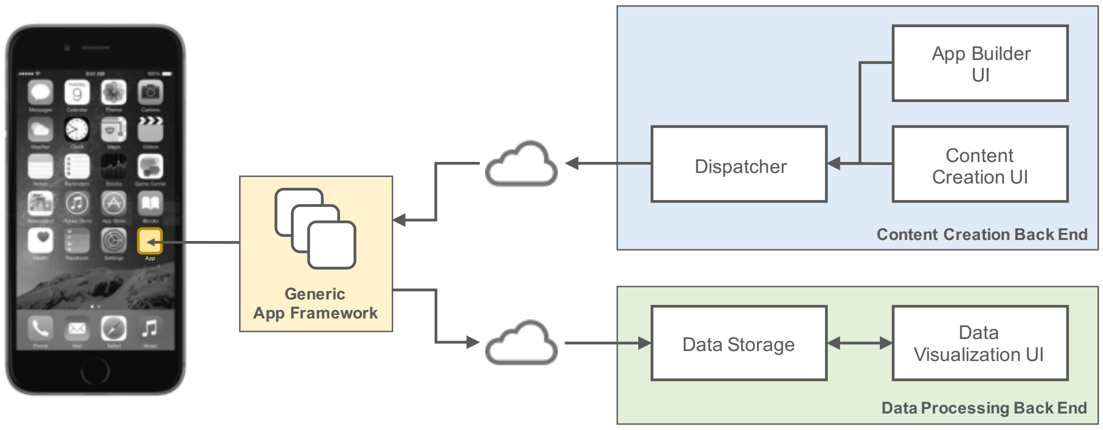


FIGURE 2.1: Overview of the basic system components

Owners of a deployment scenario work exclusively with the platform's UIs which provide role-based access to the system's administration, data visualization and content creation features. At this point it should be noted that the UIs are integrated and are only visualized as separate entities in Fig. 2.1 to illustrate the division of tasks. As a matter of fact, both the App Builder UI and the Content Creation UI are implemented within the same application – the Data Visualization UI is then included within this application in form of an iFrame.

This thesis proposes an architecture that combines both application-specific aspects and data collection aspects within the same platform. Therefore, the first main component is the Content Creation Back End's ***App Builder UI*** which allows for configuring a sensing app for the specific deployment scenario. This configuration defines the app's design, its feature set and its sensing behavior (i.e. when and where, which recording should be performed). The resulting app and sensing definitions are passed on to the back end's Dispatcher component.

The Content Creation Back End's second main component is the ***Content Creation UI*** that can be used to create contents for the features that have been assigned to the sensing app via the App Builder. Examples include geographic information to be displayed in an app's map feature or messages to be received by an app's messaging feature. The resulting content resources are again passed to the back end's Dispatcher component.

In order to allow for a quick creation of sensing apps for new deployments, all apps are instances of the ***Generic App Framework***. This means that within the scope of the app framework's existing feature set there is no need for writing any actual code. All that needs to be done in order to build an app is to initialize the framework with a unique app identifier issued by the App Builder UI. The framework then fetches the app definition resources – and for each app feature the corresponding content resources – from the Dispatcher in order to create the intended app design, feature set and sensing behavior. This fetching process takes place repeatedly at configurable time intervals to ensure that the app is always up to date. The framework has been implemented on the Apple iOS and Google Android platforms resulting in apps that can be run natively on about 97 % of all smartphones.

One of the key components in the fetching process mentioned in the previous paragraph is the Content Creation Back End's ***Dispatcher*** whose sole purpose is the delivery of the latest available resources to all sensing apps that are managed within the same back end. The delivery process is driven by a versioning approach which allows for incremental updates to app resources. This facilitates efficient app updates at all stages of a deployment and enables deployment owners to update app aspects like design, feature set or sensing behavior at any time in real-time.

As mentioned above, an app's sensing behavior can be defined by deployment owners via the App Builder UI. The recorded data is pre-processed by the app and then sent to the Data Processing Back End's ***Data Storage*** component. There, it is aggregated and processed according to the crowd sensing scenario's requirements. The results of this data analysis step can be analyzed and visualized via the ***Data Visualization UI***.

The implementation and specifics of the Generic App Framework are presented in detail in section 2.5.1. There, the methods for realizing the dynamic app configuration and the update mechanisms are also explained. Furthermore, it is shown how the framework's architecture allows for an easy extension with respect to new features. Section 2.5.2 presents and discusses the design of the the Data Processing Back End. The process of resource creation and delivery is discussed in section 2.5.3 which covers the implementation details of the Content Creation Back End. Finally, section 2.5.4 presents the communication logic between the architecture's components for the most important system functionalities.

2.4 System Deployment Walkthrough

The following section has the purpose of making the reader familiar with how the proposed Data Collection Platform is being set up and what features it offers with respect to enabling quick and efficient crowd sensing deployments. It focuses on illustrating the steps that lead to a ready-to-use crowd sensing environment, thereby demonstrating the ease of setting up the system.

Initial Setup and Administration

Setting up a crowd sensing deployment can be done solely via the platform's front end. After logging into the UI, owners of the deployment are presented with the home screen that can be seen in Fig. 2.2. On a semantic level, the system encapsulates each specific deployment in an app. Consequently, the home screen contains all apps that are associated with an instance of the Data Collection Platform. Apps are organized in accounts in order to enable multiple independent deployment owners to work with the system. Of course, owners only see those apps that are within their area of responsibility.

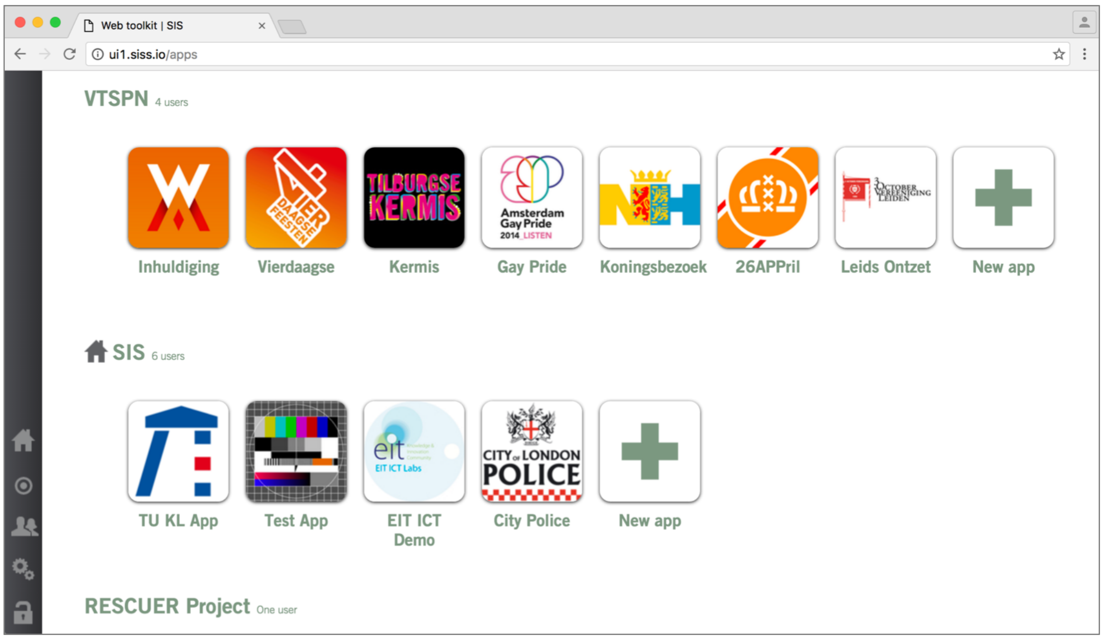


FIGURE 2.2: Screenshot of the front end's home screen

On the home screen, users can either access an app directly by clicking on its icon or they can create a new app within an account by clicking on the "New app" button. In order to create a new app, they only need to provide a name, an icon, the app's timezone and a unique identifier. The new app is then added to the home screen. The vertical bar on the left-hand side of the UI contains a context sensitive menu that's used for navigating between the UI's features. On the home screen, it only contains buttons for managing platform modules (details for that mechanism are given in section 2.5.1), user management and personal settings (e.g. for changing one's password). After selecting an app from the home screen, the users are brought to the app's dashboard (see Fig. 2.3) where they can start configuring the crowd sensing deployment.

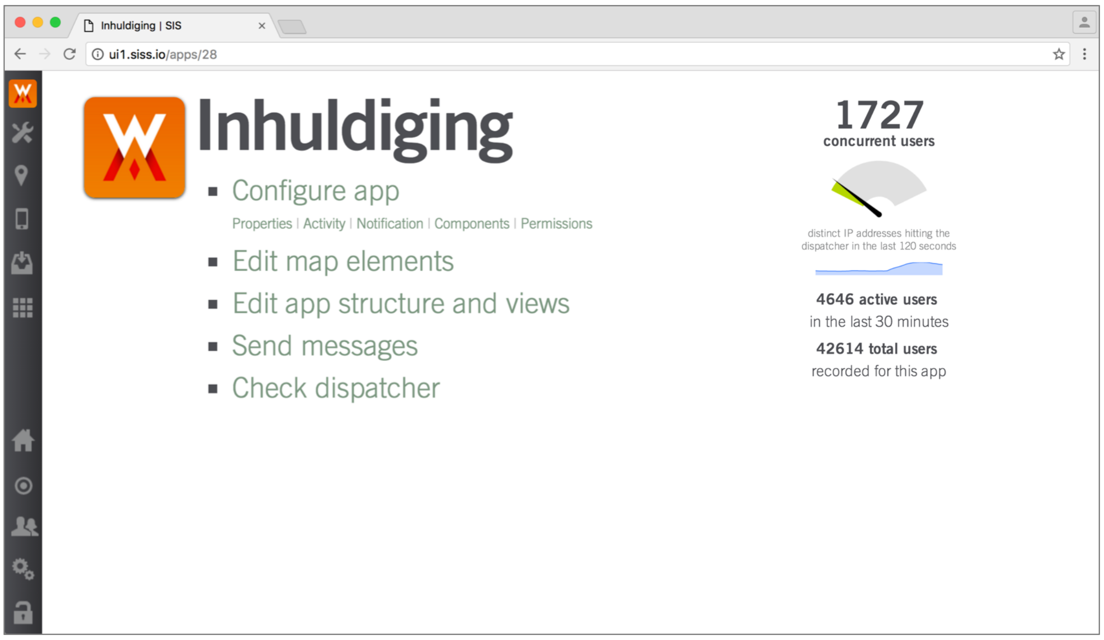
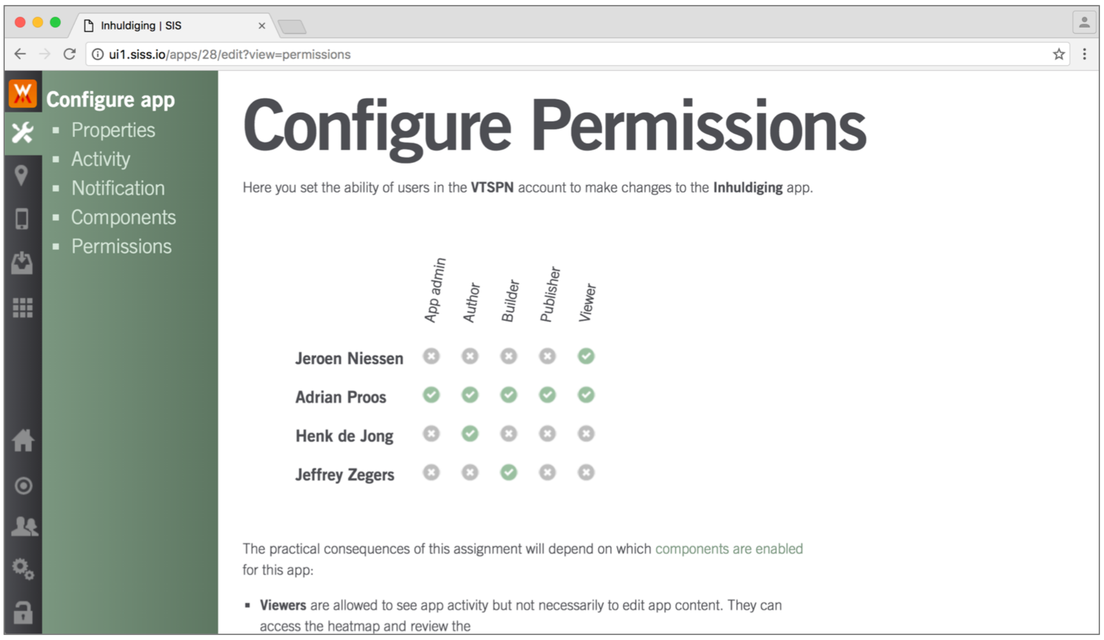


FIGURE 2.3: Screenshot of the front end's app dashboard

The app dashboard's navigation menu bar allows for accessing both basic settings (e.g. defining the activated modules in the app, defining the data recording time windows, etc.) and features like the App-Builder, Content Creation and Data Visualization UIs. The most important features are also listed as large links on the dashboard's central page – those contents do of course depend on the app's active modules. On the right-hand side, a widget measuring user activity can be seen: it shows the amount of users that are currently interacting with the system, e.g. by sending data or by requesting resources. This is done both in form of an absolute number and in form of a gauge visualizing the current load, the system is under. If the latter is reaching its maximum, the system automatically deploys a new server instance in order to be able to cope with the load. Below the load gauge, a mini chart shows the user activity during the past two minutes. At the bottom of the widget, the number of users interacting with the system during the past 30 minutes and the total number of users that have ever interacted with the system is shown.

Previously, it was already mentioned that the system provides a role-based access. Hence, not all users can access all available accounts and apps. This concept is deeply integrated into most aspects of the system and can be configured via the system's permissions screen (see Fig. 2.4). Here, different roles can be assigned to all users with access to the currently selected app.



Configure Permissions

Here you set the ability of users in the **VTSPN** account to make changes to the **Inhuldiging** app.

	App admin	Author	Builder	Publisher	Viewer
Jeroen Niessen	✗	✗	✗	✗	✓
Adrian Proos	✓	✓	✓	✓	✓
Henk de Jong	✗	✓	✗	✗	✗
Jeffrey Zegers	✗	✗	✓	✗	✗

The practical consequences of this assignment will depend on which components are enabled for this app:

- **Viewers** are allowed to see app activity but not necessarily to edit app content. They can access the heatmap and review the

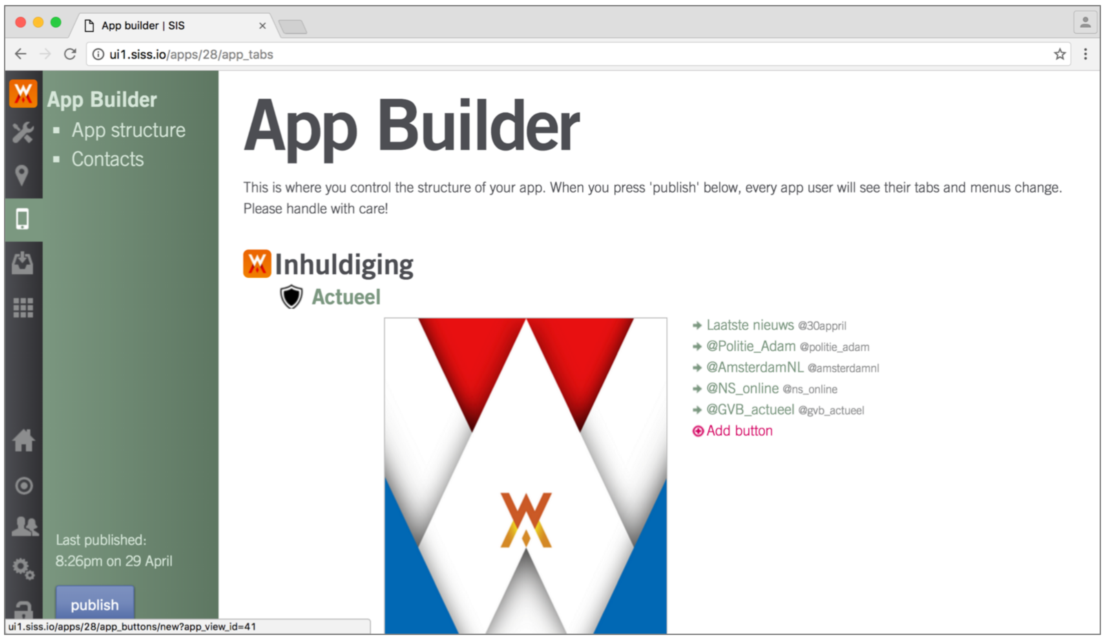
FIGURE 2.4: Screenshot of the front end's permissions screen

By default, the following roles are supported by the system:

- **App Admin** – administrators have all permissions and can also make changes to the properties of an app.
- **Viewer** – viewers can see all aspects of an app (incl. data visualizations) apart from its properties. However, they cannot edit or publish anything.
- **Author** – authors are allowed to create and edit contents for the application-specific modules of an app. They automatically have Viewer privileges.
- **Builder** – builders are allowed to change an app's feature set and its look and feel. They are not automatically granted Author or Viewer privileges.
- **Publisher** – publishers can create and send messages to the users of an app. This role is only available if the messaging module has been activated in an app.

After some initial system-users have been created and assigned to their roles, the next step in a crowd sensing deployment is usually the creation of an app which can be run on the end users' smartphones. In the previous section it was mentioned that within the proposed architecture, smartphone apps are instances of the Generic App Framework and get their structure, feature set and design properties from app definition resources.

Creating those resources is the task of the App Builder UI which can be accessed from the dashboard (see Fig. 2.5).



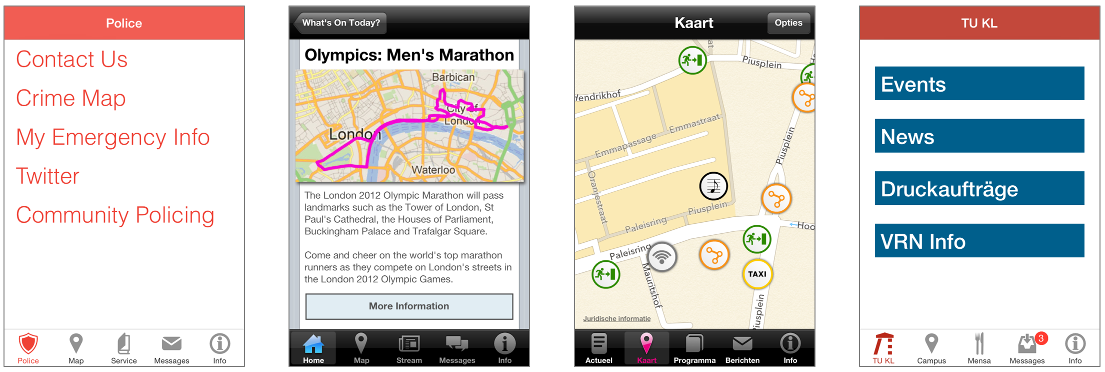
This is where you control the structure of your app. When you press 'publish' below, every app user will see their tabs and menus change. Please handle with care!

FIGURE 2.5: Screenshot of the front end's app builder screen

The app design process typically starts by defining the basic app structure which consists of a number of so-called "tabs". Each tab is a page in the app which either consists of an app module (i.e. feature) or of a menu page giving access to multiple app modules. In the above screenshot, the app's first tab is named "Actueel" (Dutch for "current") and has been configured as a menu page. The App Builder also allows for uploading icons which are then assigned to tabs. Icons are required to be colored black and have transparent backgrounds as the app framework takes care of assigning the correct colors to the icons based on the app design resource file.

If a tab has been configured to be a menu page, the users can also define if the background of that menu should consist of a solid color or if a background image should be used – in case of the above screenshot, the latter was chosen. Users can add app modules to menu pages by clicking on the "Add button"-button. Afterwards, the button's caption and the module it should trigger are set. In case a module requires start parameters (e.g. a Twitter username for the Twitter module), they can be entered in the same dialogue. In the above screenshot, the "Actueel" menu page consists of five buttons which are all linked to the Twitter module – the respective start parameters are shown in grey next to each button caption.

Once the app's structure and feature set have been finalized, they can be applied by clicking on the "publish" button. Afterwards, all running instances of the app consume the new app definition resources files[2] and re-configure themselves accordingly. Figure 2.6 shows some examples of iOS apps that have been created using the Generic App Framework.



The London 2012 Olympic Marathon will pass landmarks such as the Tower of London, St Paul's Cathedral, the Houses of Parliament, Buckingham Palace and Trafalgar Square.

Come and cheer on the world's top marathon runners as they compete on London's streets in the London 2012 Olympic Games.

FIGURE 2.6: Screenshots of apps created with the Generic App Framework. From left to right: City of London Police App, Westminster London 2012 Olympics App, Dutch Carnival App, University App of the Technical University of Kaiserslautern

During the initial setup of a deployment, system administrators (or any other technical staff for that matter) only have to start an empty instance of the Generic App Framework within their iOS or Android development IDE, enter the unique app identifier and wait until the required resources have been fetched. Once this step is completed, the app is ready for distribution and can be submitted to the respective platform's app store.

The big advantage of this approach is that apps, which already have been published, can be updated at any point in real-time without having to go through a potential app store review process. In extreme cases (e.g. during critical incidents) the structure and feature set of an app can be completely changed in order to allow for dealing with new requirements.

The set of features available to an app created with the Generic App Framework can be divided into two categories: scenario-independent and scenario-specific. In the following Table 2.1 an overview about the former is given. Chapter 3.3.4 will present those features that have been added to the framework with the goal of addressing a specific example deployment scenario.

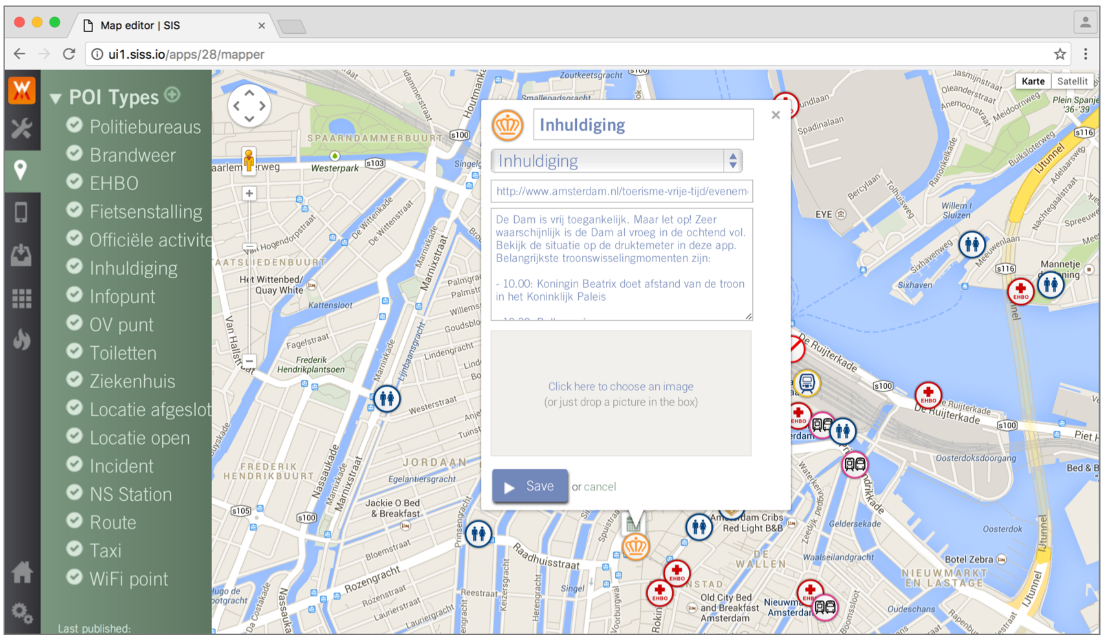
[2] Please refer to section 2.5.1 for the specifics on the format and contents of app resource files

Module Name	**Module Description**
Map	The Map Module allows for displaying points of interest (POIs) created via the Content Creation UI on a map. Upon tapping a POI, users can either compute a route from their current location to that POI or access additional information about the POI. Such information can consist of text, an image and a hyperlink.
Contact	The Contact Module presents the users with options to get in touch with entities either via call, email or by accessing a web resource. The available entities and their corresponding means for establishing contact are set via the Content Creation UI. If a user presses a button, the appropriate action (e.g. making a call or creating a new email) is performed automatically.
Messaging	The Messaging Module allows app users to receive messages from the deployment owners – this also includes location based messages, which are received only in specific locations and therefore allow for context-sensitive message delivery. The messages themselves are composed via the Content Creation UI.
Static Contents	The Static Contents Module makes large files (e.g. PDF files) available via the app. Given that those contents can potentially be of considerable size, they have to be added to the app's resource bundle at compile time – hence they can't be loaded dynamically. The module can be bound to a specific piece of content via the App Builder UI.
Web Contents	The Web Contents Module is very versatile as it allows for displaying any sort of website or online resource via an in-app browser. The module can be bound to a specific web content via the App Builder UI.
Privacy	The Privacy Modules are mandatory for each app created with the Generic App Framework and provide access to information about sensor status, legal texts and about recording activities. With respect to the latter, it is always shown when and where the next (or current) recording is taking place.

Table 2.1: Overview about scenario-independent app framework features.

Content Creation

Apart from delivering sensor data for the crowd sensing deployment, the main task of the architecture's smartphone app is to provide its users with some sort of motivation to keep the app installed on their phones. Such motivation primarily comes from the contents provided by the modules used in the app – e.g. in form of information or useful/entertaining features. While some modules may not require the deployment owners to manually set their contents at all (e.g. in case of gamification modules), others are quickly set up via start parameters in the App Builder UI during the app creation process. The third group of app modules are those requiring more complex contents (e.g. geographic information in case of the Map Module) which are set up via the Content Creation UI. In the following, it is shown how this UI is operated by demonstrating the content creation process using the example of two representative app modules, which are both relevant to smart city scenarios.



Inhuldiging

Inhuldiging

http://www.amsterdam.nl/toerisme-vrije-tijd/evenem

De Dam is vrij toegankelijk. Maar let op! Zeer waarschijnlijk is de Dam al vroeg in de ochtend vol. Bekijk de situatie op de druktemeter in deze app. Belangrijkste troonswisselingmomenten zijn:

- 10.00: Koningin Beatrix doet afstand van de troon in het Koninklijk Paleis

FIGURE 2.7: Screenshot of the map editor in the content creation UI

The first example is the creation of geographic information for the Map Module (see Fig. 2.7 for reference). A new POI can be created by simply clicking on the desired location on the map which causes a dialog window to pop up. Here, the user must define at least a name for the POI and assign it to a category via the dropdown menu, which also defines the POI icon. POI categories are directly defined in the green menu bar using the "+" button next to the "POI Types" heading. The rest of the POI information is optional. Any amount of text can be added via the large text box – e.g. to provide some background information on the POI. Also, a hyperlink can be added to the POI using the text field just below the POI category dropdown menu. Finally, an image can be attached to the POI – for example by dropping an image file in the grey image box near the bottom of the dialog. After all the information has been entered, the POI is created by clicking on the "save button".

Once all required map contents have been added, they are published by clicking on the "publish" button (which is not visible in Fig. 2.7 as it is below the expanded list of POI categories). This button causes the Content Creation UI to assemble the map module resource file and pass it on the the Dispatcher component, where it can be consumed by apps. Again, all map information can be edited and re-published at any time to make sure that the Map Module is always up to date.

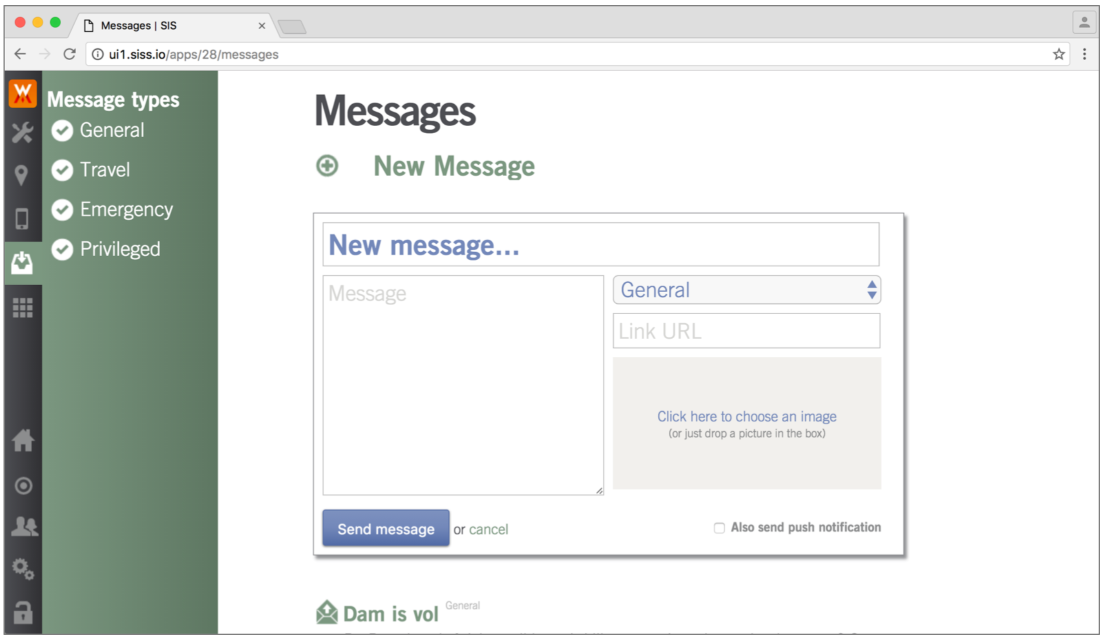


FIGURE 2.8: Screenshot of the message editor in the content creation UI

As a second example for content creation, the Message Editor was chosen (see Fig. 2.8 for reference). It is being used to send broadcast messages to all users of an app. After clicking on the "New Message" button, the message composer is opened where the user can enter a message title and the message itself. Furthermore, a message channel has to be set via the dropdown menu on the left hand side. The Message Module offers a channel for general messages, one for emergency messages and one for travel/transportation messages. Those three channels are simply a way to organize messages into separate inboxes within the app's Message Module. The fourth channel for privileged messages however, is invisible to most end users. In order for it to appear in the app, the user has to perform a special gesture on the smartphone's touch screen and enter a PIN afterwards. The purpose of this channel is to create the means for distributing messages only to a select pool of insiders (e.g. staff of the deployment owner) via the app. Finally, hyperlinks and images can be added to a message just like they are added to a POI. In contrast to the Map Editor, the Message Editor creates a new message resource file for the dispatcher immediately after the "Send Message" button has been clicked in order to ensure that messages are delivered as quickly as possible. In case of very relevant messages, there's also the option to send a message via a push notification by enabling the respective check box next to the "Send Message" button.

If a message is only intended for people who are located in a certain area, the message composer can also be opened from the Map Editor. Here, the user is first asked to draw a polygon on the map, thereby defining the region where the message should be received. Afterwards, the standard message composer window pops up in the same layout as in case of broadcast messages.

Managing Data Recording Sessions

Once the crowd sensing app has been submitted to the app stores and has been downloaded by a reasonable amount of users, the final step of the deployment can take place: the definition of data recording sessions. For reasons of protecting the end users' privacy, the proposed Data Collection Platform only allows for data recordings within specific geographic boundaries (also: "data recording zone") during specific time windows.

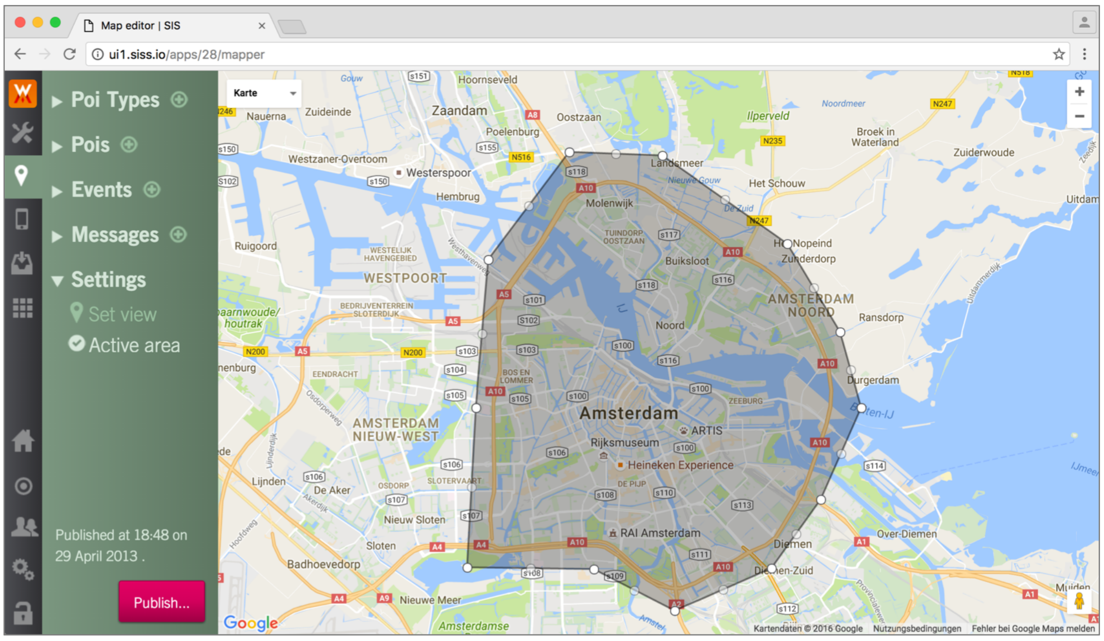
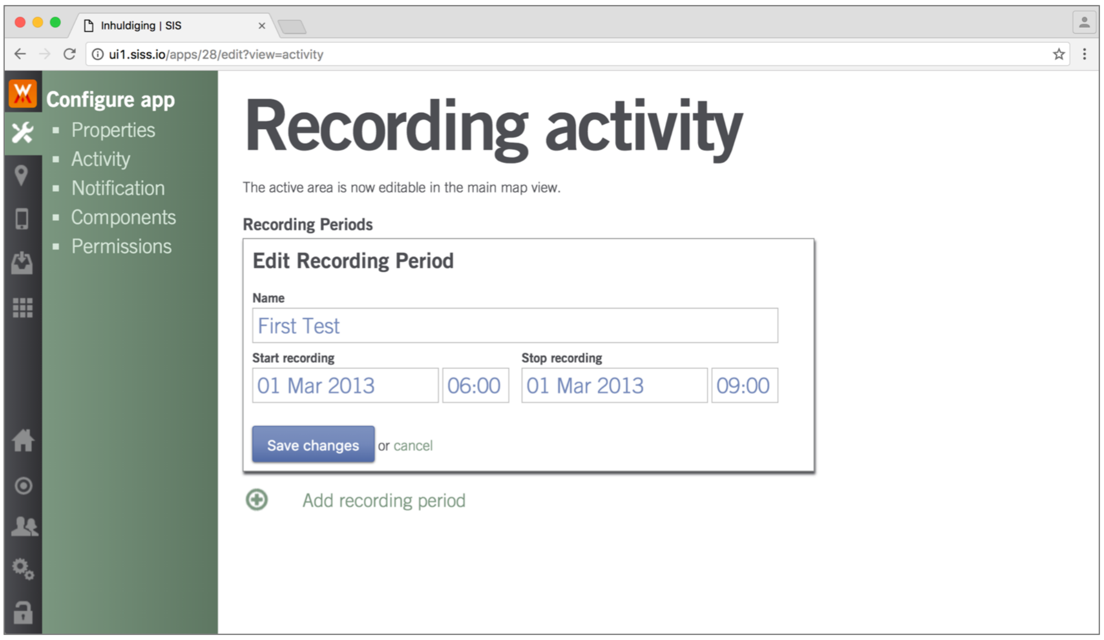


FIGURE 2.9: Screenshot of the mapping editor while defining the data recording zone

The data recording zone is also created via the Map Editor. Here, the user can define or edit it by clicking on the "Active Area" link within in the map settings (see Fig. 2.9). Similarly to creating the zone of a location based message, the user only needs to draw a polygon on the map in order to define the recording zone. A click on the "Publish" button updates the sensing resource file and sends it to the Dispatcher. The time windows, within which data is recorded from all users who are currently in the data recording zone, are defined in the "Recording Activity" menu of the app's main configuration page. Here, the user can create any amount of recording periods by specifying start and end dates/times for each recording period. Again, a click on the "Save Changes" button sends an updated sensing resource file to the Dispatcher.



Recording activity

The active area is now editable in the main map view.

Recording Periods

Edit Recording Period

Name	First Test		
Start recording	01 Mar 2013 06:00	Stop recording	01 Mar 2013 09:00

FIGURE 2.10: Screenshot of the recording activity configuration page

Of course, the actual processing and visualization of the data recorded with the proposed platform depends heavily on the concrete scenario of the deployment. Because of that, chapter 2.5 presents in detail the measures that have been taken to enable a flexible implementation of data recording and processing services within each platform component in order to allow for a wide range of applications.

Within the course of this thesis, the scenario of analyzing crowd density with the help of crowd sensed location data has been examined (see chapter 3 for details). Therefore, Fig. 2.11 shows an example of how a rendering of such data within the Data Visualization UI looks like for that particular scenario. It can be seen that crowd density information has been transformed into a heat map which is then superimposed on top of a geographic map. At this point, the purpose of this figure is merely to demonstrate the degree of the proposed Data Collection Platform's integration – a single front end can be used to create crowd sensing apps, to update their contents, to manage their recording activity and to visualize the recorded data.

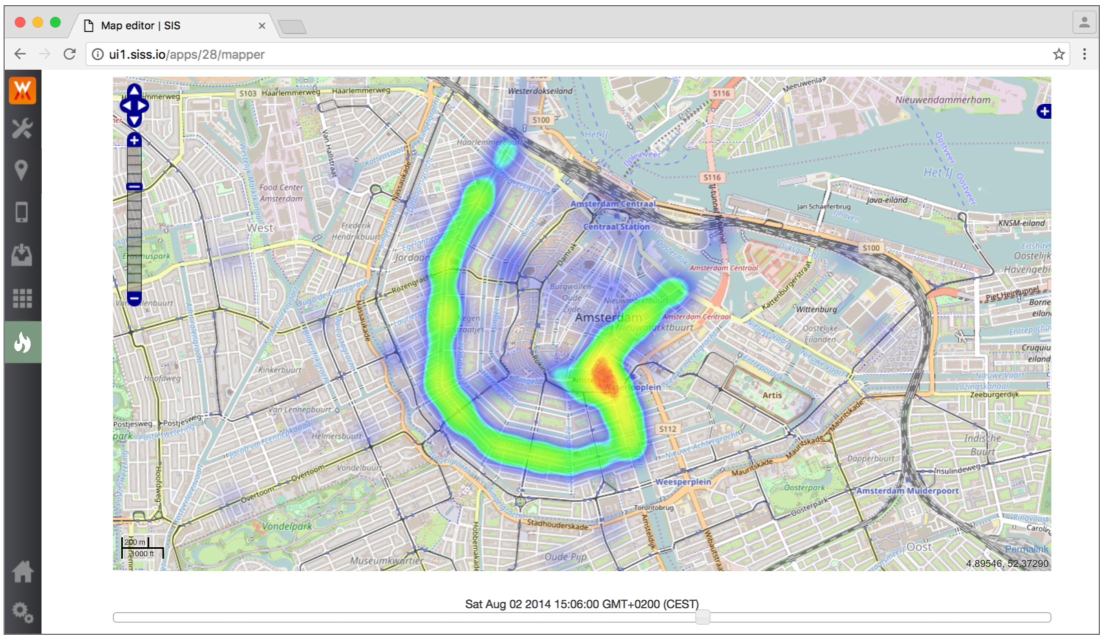

Map editor | SIS
ui1.siss.io/apps/28/mapper
Amsterdam
Sat Aug 02 2014 15:06:00 GMT+0200 (CEST)

FIGURE 2.11: Screenshot of a crowd sensing data visualization

2.5 Architecture Details and Implementation

The following sections present the most important details of the proposed architecture and provide insights on why certain design decisions have been made. At the beginning of each section, a "General Considerations" sub-section puts the component into context and explains its underlying concept. Each of the subsequent sub-sections then highlight one particular aspect of that component.

2.5.1 Generic App Framework

The main challenges with respect to implementing a Generic App Framework for crowd sensing applications are:

1. the ease of creating new apps based on that framework,
2. keeping the contents of those apps up to date, and
3. allowing for extensions of the app framework's feature set.

While the general idea of dynamically creating and updating apps by means of resource files created via the system's front end has already been introduced in the previous sections, the following sub-sections are presenting the implementation details.

General Considerations

Nowadays, a multitude of approaches for implementing smartphone apps is available. However, each one of them has its specific advantages and downsides. [29] presents an overview about the most common app development techniques and compares them based on a literature review. In their work, the authors present the following options:

- **Native Apps** are developed using the smartphone platform's standard development tools and programming languages – e.g. Java in case of Android and Objective-C in case of iOS. They offer the richest user experience, fastest performance, most efficient source code and give full access to the underlying smartphone hardware. The downside is that implementing native apps is often considered as more difficult as this approach requires a higher level of experience and platform-specific know-how. Also, code that has been written for one platform cannot be re-used for another platform.
- **Web Apps** on the other hand are the exact opposite of native apps. They are developed using widespread technologies such as HTML and JavaScript which are known to a large number of developers. The main disadvantages of this approach however, are the limited access to the device's hardware, the overhead required to render the UI and the extra time required for downloading the web pages from the internet. In addition to that, it is very hard to mimic the individual UI details of a specific platform with web apps.
- **Hybrid Apps** are trying to combine the strengths of native apps and web apps with the most popular framework being *PhoneGap*. Here, the app's logic is again implemented in HTML5 and JavaScript but this time it is running in a thin native container. Nevertheless, the source code is still executed by a browser which leads to disadvantages in terms of performance. To a certain degree, access to the device hardware can be gained through specialized APIs. However, if a near-native look look and feel of the resulting app is the goal, specific development libraries such as *JQuery* are required.
- **Interpreted Apps** are automatically producing native code to implement the user interface of an app. The application logic itself is implemented using programming languages and technologies such as Java, Ruby and XML. Besides the native look and feel of the resulting apps, a faster performance than web apps or hybrid apps is the main upside of this development approach. However, the big disadvantage is that the developer is completely dependent on the vendor of the software development environment – i.e. only platform features supported by the environment can be used inside apps. This can be especially limiting when new versions of a

smartphone platform are released and the development environment hasn't been updated to support those, yet. An example for this approach is the *Appcelerator Platform*.

- **Generated Apps** are compiled like native apps and a specific version of the app is created for each platform. In principle, this leads to the same advantages that characterize native apps. However, the development environments for generated apps (e.g. *Applause* or *iPhoncal*) are following a model-driven software development (MDSD) process, where the problem domain is described in a model which is then used to generate the native code. Apart from not offering the same degree of freedom as native programming languages, the used model languages are far from being common practice amongst developers which leads to a relatively small pool of developers familiar with the subject.

In summary, the authors of [29] come to the conclusion that *"there is no best solution regarding the cross-platform development approach that should be chosen for a mobile application in general"*. Taking this assessment as a basis and considering the fact that crowd sensing apps by definition require the best possible access to a device's hardware features (i.e. sensors), the decision was made to implement the Generic App Framework in form of native apps. Therefore, two separate versions of the framework for iOS and Android have been developed which offer the best possible performance, access to sensors and user experience. It is the author's opinion that those advantages outweigh the downside of having to maintain two code bases – especially given the fact that smart city deployments are usually run in a professional environment where maximum performance is required.

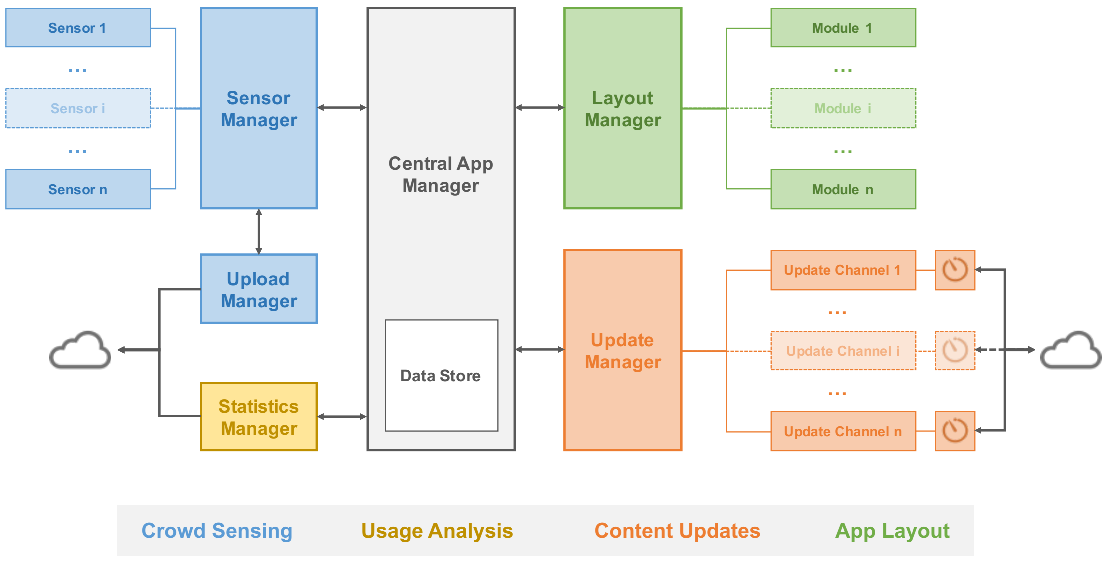


FIGURE 2.12: Overview about the general architecture of the app framework

Figure 2.12 presents the high-level architecture of the Generic App Framework. The core of each app is the *Central App Manager* which controls the app's lifecycle and manages data storage. It also facilitates communication between all other components by offering a notification service via a standard publisher-subscriber-pattern. While the *Layout Manager* is the component responsible for arranging the app with respect to its feature set and design, the *Update Manager*'s main task is to keep all app resources (e.g. app design definitions, sensing behavior definitions, module contents, etc.) up to date. It therefore communicates with the system's back end via the internet to gather the latest versions of those resources. The tasks of recording and uploading sensor data whenever the user is participating in a crowd sensing session, are the responsibility of the *Sensor Manager*. Finally, the *Statistics Manager* provides the means to record and upload usage statistics giving insights into how users are actually interacting with the app.

Keeping App Resources up to date

The Update Manager is responsible for keeping all aspects of an app up to date. This ranges from contents that may be required for certain app modules (e.g. points of interest to be displayed within the map module) and goes as far as the definition of the app structure with respect to its features and their placement within menus. All these required app resources are provided in form of JSON files (**J**ava**S**cript **O**bject **N**otation) by the system's cloud-based Content Creation Back End. The Update Manager retrieves the resources via its update channels, with each channel being responsible for one type of resource only. Update channels are driven by a timer controlling the frequency with which the channel contacts the back end for the purpose of checking for updates to its resource.

The Update Manager is initialized by the Central App Manager during the app's startup routine. At this point, the update channels are also initialized by the Update Manager with their respective update intervals. Towards the end of the app's startup routine, the Central App Manager starts all available update channels.

Now, if the framework should be extended by new modules which may require their own app resources to be fetched via the internet, a developer can do this by simply subclassing the framework's `UpdateChannel` object. Within that new object, the methods for creating the HTTP-request (`setupRequest`) and for handling the server's response (`updateAppDataWithRequestBody`) can be overwritten for customizing the behavior of the new channel.

If an update channel's timer fires, an HTTP-request is created by the `setupRequest` method. A very important aspect at this stage is the handling of the app resource's current version. This is handled on two layers: at the HTTP request layer, it is done by using the `If-None-Match` HTTP header which makes the request optional. This means that the back end will send the online resource, with a 200 status, *only if it doesn't have an ETag matching the one given in the header*. When the condition is not met (i.e. the ETags match), the back end simply returns the HTTP status code 304 (Not Modified). Therefore, the `setupRequest` method sets the `If-None-Match` header to the ETag that was returned by the back end with the current version of the app resource. After the header has been set, the request is sent. While the URL for this request may vary for custom update channels, the ones that are part of the core framework are all formatted after the following schema: `https://serverAddress/platform/apiVersion/appID/channelName` where `serverAddress` contains the address where the back end can be reached and `platform` contains either `ios` or `android` depending on the specific device.

When the back end responds to an HTTP request, the update channel's `requestFinished` method is called. Here, it is first checked, if the response code 200 is present (i.e. if a new resource has been sent). If that is the case, the response string is passed to the `updateAppDataWithRequestBody` method where it's parsed and converted into a JSON object. The resulting new resource version is then stored in the Central App Manager's Data Store, which is a very simple key/value-based storage mechanism. Finally, the update channel stores the current resource's ETag for the next update request and sends a notification to all objects that subscribed as listeners to the update channel's *"new data available"* event. At this point, the new resource is consumed by all relevant objects, which then act upon it – an app module for example would now update its contents.

```json
{
  "revisionnumber": 0,
  "items": [
    { "content": "101",
      "highlighted": false,
      "type": "phone number",
      "name": "Non-Emergency Crime" },
    {
      "content": "999",
      "highlighted": true,
      "type": "phone number",
      "name": "Emergency Call" },
    {
      "content": "postmaster@cityoflondon.police.uk",
      "highlighted": false,
      "type": "email address",
      "name": "General E-Mail Enquiries" }
  ]
}
```

Listing 2.1: Example of an app resource JSON

Listing 2.1 shows an example of a JSON object received via the contact module's update channel. While the majority of its content is not of relevance at this point, the `revisionnumber` attribute in line 2 is where the second layer of version control is realized. Each JSON object returned from the back end has this attribute which contains the resource's version number in case the module itself needs to react on it. Most standard app modules simply replace their old contents with the new version. More complex modules however, have an incremental update mechanism (e.g. the broadcast message module) where only the latest changes or additions to a resource are sent in order to minimize the amount of transmitted data for each update. Those modules need the revision number in order to keep track of their content's order.

Dynamically configuring the App

The main contribution of the proposed Generic App Framework is its ability to quickly create new apps for a given scenario without any actual implementation efforts. It was already mentioned that this is accomplished by encapsulating all the information about an app's structure and its look and feel in app resource JSONs which are received by the Update Manager. Controlling this design process is the task of the Layout Manager: it reads the relevant JSONs, lays out the app accordingly and assigns the required app modules to its corresponding UI element.

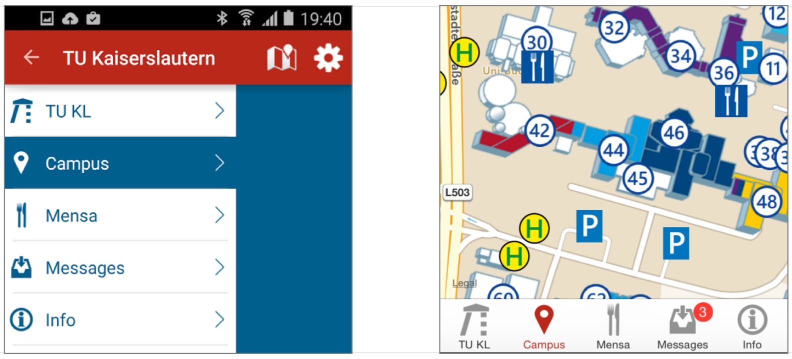


FIGURE 2.13: A comparison of an Android and an iOS app: both apps use the same structure definition but render them according to their platform's design guidelines.

Another goal of the Generic App Framework is to create apps which provide a platform-specific experience for its users, i.e. the apps need to adhere to the respective platform's UI guidelines. For that reason, iOS apps created with the framework use a so-called tab-bar as main navigation paradigm whereas Android apps use the established slide-out navigation menu. This results in apps where users "feel right at home". Figure 2.13 shows a comparison between an Android and an iOS app – both apps are using the same design resources but interpret them differently in order to provide a platform-specific experience.

The underlying principle of the dynamic layout process is that all apps created with the framework are practically identical with respect to their code base. Each app is compiled with all possible app modules on board. However, the decision, which modules are actually used and how they are designed, is only made at run-time. Therefore, the Layout Manager is the first object that's created by the Central App Manager during an app's startup routine. After having been initialized, the Layout Manager gets the app structure JSON and the app design JSON from the Central App Manager's Data Store and evaluates them in order to lay out the app properly. Listing 2.2 shows an excerpt of an app structure JSON.

```
{
  "revisionnumber": 0,
  "splashscreenpath": "https://[url]/TabBarIcon-Police.png",
  "starttab": "0",
  "usespushmessaging": true,
  "tabinfos": [
    {
      "title": "Police",
      "iconpath": "https://[url]/TabBarIcon-Police.png",
      "mainview": {
        "backgroundpath": "https://[url]/LCP_Background.jpg",
        "mainbuttoninfos": [
          {
            "title": "My Emergency Info",
            "onlinedependency": "NO",
            "viewcontroller": "EmergencyInfoViewController",
            "startstring": "" },
          {
            "title": "Twitter",
            "onlinedependency": "YES",
            "viewcontroller": "TwitterViewController",
            "startstring": "@citypolice" },
          // <...>
        ]
      }
    },
    {
      "title": "Map",
      "iconpath": "http://[url]/TabBarIcon1-Maps.png",
      "directcontroller": "MapsViewController",
      "startstring": "" },
    // <...>
  ]
}
```

Listing 2.2: Excerpt of an app structure JSON

In line 3, the path to the splash screen, which is shown for one or two seconds while the app is loading, is defined. It is worth mentioning, that all online resources (i.e. images and icons) are downloaded and stored locally by the Layout Manager after a new version of an app definition JSON has been received from the Update Manager. Hence, there is no need to download anything during app startup, which makes the whole process very quick. The `starttab` attribute defines the main menu which should initially be shown after the app has loaded – in case of the above example it would be the first item. If the `usespushmessaging` attribute in line 5 is set, the app registers with the operating system for push notifications.

Starting at line 6, the actual structure of the app is defined. The `tabinfos` array contains the information about each of the app's main menus (also referred to as "tabs"). Each item within this array corresponds to a tab on iOS or a navigation menu item on Android (see Fig. 2.13 for clarification). Each main menu is assigned a title and an icon (e.g. lines 8 and 9). With respect to the latter, the structure JSON only defines the icon itself; how that icon is colored is defined in the design JSON (see Listing 2.3).

Next, the exact type of main menu is defined. Generally, there are two options available: assigning an app module directly to the main menu or having the main menu show a list of buttons which open app modules when they are pressed. In case of the former, the `directcontroller` attribute is set (see line 30). It contains the name of the app module's class. During the app's startup process, the module is then initialized by creating the object in a standard `classForName`-fashion. As app modules might need initialization parameters, the `startstring` attribute is present – if that attribute is left empty, the module doesn't need any information in order to start up properly.

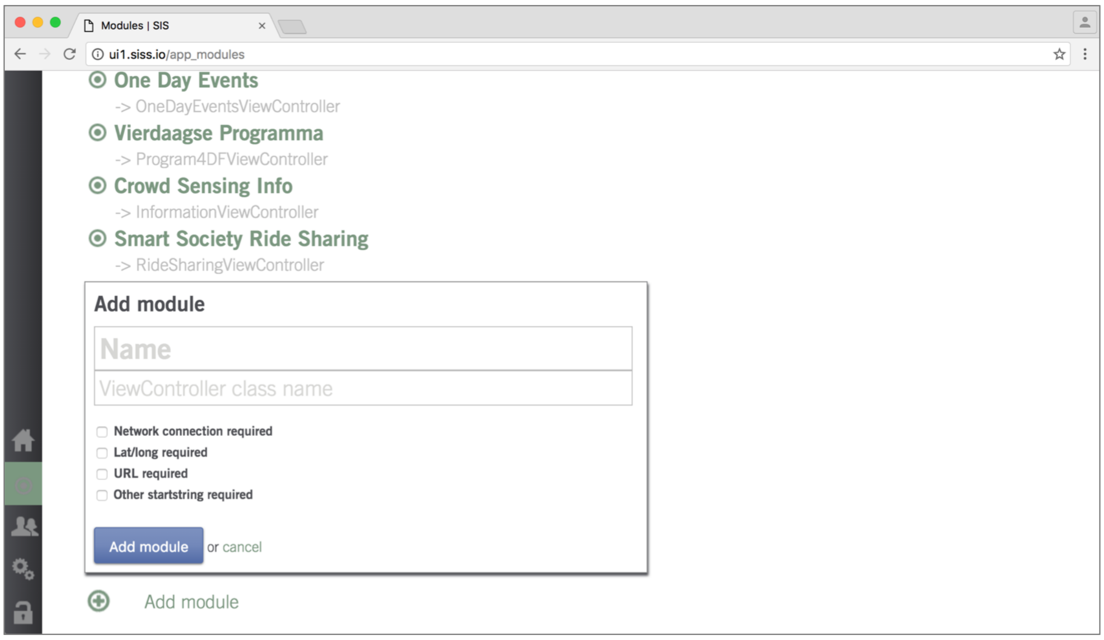
If a main menu is defined to show a list of buttons, the "mainView" attribute is set (see line 10). This attribute contains an array of items each representing a menu button. Again, each button has a title (set e.g. in line 14). In addition to that, the structure JSON defines if the app module assigned to that button requires online connectivity via the `onlinedependency` attribute. If that attribute is set and the device is offline, the respective button is disabled and an information message is displayed if users are touching it. Assigning the actual app module to a button takes place in the same way as in the previous case with the `viewcontroller` attribute specifying the module's class name and the `startstring` attribute containing the initialization parameter (e.g. a Twitter Handle in case of the Twitter module defined in lines 21 and 22).

As apps running on iOS and on Android are using different navigation paradigms, there are also separate design JSONs for each platform. The following Listing 2.3 shows an example of a design JSON for iOS. Given the fact that the attribute names are self-explanatory (e.g. the aforementioned tab-bar icon color is set in lines 6-9), they are not further elaborated on at this point.

```
1  {
2    "appdelegate.tabbar.tintcolor.r": 255,
3    "appdelegate.tabbar.tintcolor.g": 255,
4    "appdelegate.tabbar.tintcolor.b": 255,
5    "appdelegate.tabbar.tintcolor.a": 1,
6    "appdelegate.tabbar.imagetintcolor.r": 239,
7    "appdelegate.tabbar.imagetintcolor.g": 65,
8    "appdelegate.tabbar.imagetintcolor.b": 53,
9    "appdelegate.tabbar.imagetintcolor.a": 1,
10   "maintabbar.navbar.tintcolor.r": 239,
11   "maintabbar.navbar.tintcolor.g": 65,
12   "maintabbar.navbar.tintcolor.b": 53,
13   "maintabbar.navbar.tintcolor.a": 1,
14   "maintabbar.navbar.textcolor.r": 255,
15   "maintabbar.navbar.textcolor.g": 255,
16   "maintabbar.navbar.textcolor.b": 255,
17   "maintabbar.navbar.textcolor.a": 1,
18   "maintabbar.tabbaritem.textcolor.normal.r": 45,
19   "maintabbar.tabbaritem.textcolor.normal.g": 45,
20   "maintabbar.tabbaritem.textcolor.normal.b": 45,
21   "maintabbar.tabbaritem.textcolor.normal.a": 1,
22   "maintabbar.tabbaritem.textcolor.selected.r": 239,
23   "maintabbar.tabbaritem.textcolor.selected.g": 65,
24   "maintabbar.tabbaritem.textcolor.selected.b": 53,
25   "maintabbar.tabbaritem.textcolor.selected.a": 1,
26   "maintabbar.standardbutton.width": 290,
27   "maintabbar.standardbutton.height": 40,
28   "maintabbar.standardbutton.verticalgap": 12,
29   "maintabbar.standardbutton.offsettop": 50,
30   "custombutton.titlecolor.highlighted.r": 0,
31   "custombutton.titlecolor.highlighted.g": 0,
32   "custombutton.titlecolor.highlighted.b": 0,
33   "custombutton.titlecolor.highlighted.a": 1,
34   "custombutton.titlecolor.normal.r": 239,
35   "custombutton.titlecolor.normal.g": 65,
36   "custombutton.titlecolor.normal.b": 53,
37   "custombutton.titlecolor.normal.a": 1,
38   "custombutton.fontsize": 30,
39   "custombutton.horizontalalignment": 1,
40   "custombutton.backgroundimage": "",
41   "custombutton.fontname": "Helvetica-Light"
42 }
```

Listing 2.3: Example of an app design JSON for the iOS platform

Extending the Generic App Framework with new modules is very easy. The developer only has to implement one interface which defines the name of the module's initialization method (`initWithStartString`) – apart from having to stick to the framework's navigation paradigm, there are no further restrictions. If the new module has contents that need to be updated via a remote server, an update channel can be added as described in the previous section.



Modules | SIS

ui1.siss.io/app_modules

One Day Events
-> OneDayEventsViewController

Vierdaagse Programma
-> Program4DFViewController

Crowd Sensing Info
-> InformationViewController

Smart Society Ride Sharing
-> RideSharingViewController

Add module

Name

ViewController class name

- [ ] Network connection required
- [ ] Lat/long required
- [ ] URL required
- [ ] Other startstring required

Add module or cancel

Add module

FIGURE 2.14: Screenshot of the front end's app modules screen

The last step of this extension process is to register the new module with the cloud-based administration UI. This can be done via the "App Modules" menu which can be found on the front end's home screen (see Fig. 2.14). Here, the user has to enter the new module's name and its corresponding class name. There's also the option to specify that the module requires internet connectivity or a specific initialization parameter. Once all the information has been entered, the user clicks on the "Add Module" button in order to make the new module available to the App Builder UI.

Gathering Sensor Data

From a crowd sensing perspective, the Sensor Manager is the most important component of the Generic App Framework as its responsibilities are the recording and uploading of sensor data while the user is participating in a crowd sensing session. Again, the Sensor Manager is initialized by the Central App Manager in the course of the app's startup process. During its initialization, the Sensor manager first registers as subscriber for notifications related to new crowd sensing sessions. Afterwards, all available sensor objects are initialized and stored in a so-called sensor-array. Finally, the Upload Manager is created which causes this object to initialize an upload queue.

```
 1 {
 2   "revisionnumber": 0,
 3   "geozone": [
 4     {
 5       "longitude": -12.12890625,
 6       "latitude": 59.977005492196 },
 7     {
 8       "longitude": 26.3671875,
 9       "latitude": 60.75915950226991 },
10     {
11       "longitude": 27.59765625,
12       "latitude": 37.300275281344298 },
13     {
14       "longitude": -9.84375,
15       "latitude": 38.410558250946089 }
16   ],
17   "activedays": [
18     {
19       "date": "2012-08-24",
20       "from": "09:00",
21       "to": "20:00" }
22   ]
23 }
```

Listing 2.4: Example of a sensing schedule JSON

The app's sensing behavior is controlled by the sensing schedule JSON received via the Update Manager – Listing 2.4 shows an example of such a schedule. It can be seen that a sensing session is defined by two items. Firstly, by a geographic zone (the `geozone` attribute in line 3) which is described by a set of coordinates forming a polygon. The second part of the sensing session definition is a schedule which can consist of an arbitrary number of recording time frames (the `activedays` attribute in line 17).

If one of the time frames of the sensing session definition should become active, the Sensing Manager is starting the Upload Manager which causes a repeating upload timer to be initialized with the app's upload frequency. Furthermore, the GPS sensor object is started in order to recognize when the user is inside the geographic zone defining the crowd sensing area. The GPS sensor's data is sent to the Sensor Manager's `handleSensorData` method which determines the type of the incoming data and forwards it to the responsible handler method – for GPS data, this would be the `handleGpsData` method. Within that method, a utility service is checking whether the user's current location is within the crowd sensing area. Should that be the case for the very first time in a sensing session, all objects within the sensor-array are started which causes the associated sensors to start recording data. If the user leaves the crowd sensing area, all objects within the sensor-array are stopped which leads to all sensors ceasing their operation.

The data recorded by the device's sensors is sent to the Sensor Manager's handling methods as described above. Within those methods the data can be pre-processed and wrapped in a sensor-specific data dictionary. Afterwards, the data is passed on to the Upload Manager's `handleSensorData` method where further upload-relevant attributes are added to the data dictionary. Subsequently, the data is put into the upload queue.

Every time the upload timer expires, the contents of the upload queue are passed to a separate upload task. This task is then put into a task queue which is being processed on a first-in-first-out basis. The main part of the upload task is the creation of the contents for the HTTP POST request within which the data is sent to the Data Processing Back End. For this, the data is parsed into a JSON object which is serialized in a URL-safe encoding. Furthermore, the app's user ID, the app ID and the current time stamp are made part of the request's POST body. If the POST request can't be sent successfully, the data is put again into the upload queue so that it may be uploaded during the next upload cycle.

```
// HTTP Headers:
  "Accept" = "text/plaind"
  "Content-type" = "application/x-www-form-urlencoded"

// POST Body:
aid=222&uid=358665CF-74F0-42D7-A72C-D278199443E7&ut=1366635781.533519&
    data=[
  {
    "lo": -0.09401679039001465,
    "la": 51.51218128077621,
    "t": 1366635722.142807,
    "p": 5,
    "d": 1,
    "v": -1,
    "s": 1,
    "b": -1 }
]
```

LISTING 2.5: Example of an HTTP POST request sent out by the Upload Manager

Listing 2.5 shows an example for a POST request sent by the Upload Manager to the back end. Here, the `aid` parameter contains the app ID as it is set in the administration UI; the `uid` parameter contains the app's user ID and `ut` corresponds to the upload timestamp. In this example, the uploaded data only consists of one GPS data packet. Within that packet, `lo` and `la` correspond to the GPS fix' longitude and latitude values while `t` contains the data's creation timestamp. The parameter `p` stands for the GPS fix' precision in meters. `v` and `b` stand for the user's velocity and bearing at that particular time – the fact that both values are -1 indicates that this information was not available. Finally, `d` and `s` stand for "device ID" and "sensor ID" – both values that are only required for routing the data packets within the back end logic.

If the Generic App Framework should be extended by new sensors, developers only need to ensure that they implement the framework's common sensor interface defining the `start` and `stop` methods in the new sensor object. Furthermore, an additional handler method for the new sensor type must be added to the Sensor Manager and registered with its general `handleSensorData` method. If those three steps are followed, the new sensor data is part of all crowd sensing sessions performed with the Generic App Framework.

Gathering Usage Statistics

The Generic App Framework offers the means to record usage statistics and upload them to the Data Processing Back End in regular intervals. By default, the framework logs the usage of each app module to enable deployment owners to gain insights into how their app is being used. This knowledge can then be leveraged to adjust the feature set or layout of the app accordingly.

Usage logs are created automatically by the base class of each app module. Whenever a module is accessed, the Statistics Manager's `addItemToStatistics` method is called. This method starts a 3-second timer – if the timer expires, a usage log for the app module is created. The reasoning behind this approach is that users sometimes access a module by accident – the 3-second threshold on the other hand ensures fairly reliably that only "real" usages of a module are recorded. If the user opens another module within those three seconds, the timer is re-initialized with the current module's information.

Of course, the Statistics Manager can be expanded by developers by adding an additional log method to the Statistics Manager. In theory, each module can create its own logs to provide deployment owners with further insights. Listing 2.6 shows an example of a statistics log containing one usage log (lines 2-6) and one POI access log created by the map module (lines 8-17).

```
 1  {
 2    "logtype" : "feature",
 3    "item" : "Karte",
 4    "latitude" : 48.56041099880634,
 5    "longitude" : 13.45266444440555,
 6    "timestamp" : "2012-05-03 17:05:46" },
 7  {
 8    "logtype" : "viewedpoiinfo",
 9    "poiname" : "U3/U2 Volkstheater",
10    "poicategory" : "U-Bahn Stationen",
11    "poicoordinate" : {
12      "latitude" : 48.20509857993194,
13      "longitude" : 16.35802030563354},
14    "pointofrequest" : {
15      "latitude" : 48.56041099880634,
16      "longitude" : 13.45266444440555},
17    "timestamp" : "2012-05-03 17:05:28" }
```

Listing 2.6: Example of a statistics log

The `logtype` attribute allows for distinguishing between the different log types. Within the usage log, it can be seen *where* (lines 4 and 5) and *when* (line 6) *which feature* (line 3) has been used. The map module's POI access log on the other hand records *where* (lines 15 and 16) and *when* (line 17) *which POI* (line 9) of *what type* (line 10) has been viewed. In addition to that, the *location of the POI* (lines 12 and 13) is recorded.

The process of sending out statistics data follows the same principle as the sending of sensing data (please see the previous sub-section for details). However, in order to protect the end users' privacy, the app's user ID is not sent to the back end as part of the HTTP POST request. While this user ID is of course anonymous (please see the next section for details), that decision was made out of ethical reasons in order to prevent deployment owners from being able to link crowd sensing data and usage statistics data.

Privacy Considerations

It was already mentioned in the introductory chapter that one of the most important aspects of large scale crowd sensing systems is the preservation of the end users' privacy. If users lose confidence in a crowd sensing solution, they are very likely to stop using it, which in turn can be an existential problem for the whole deployment as the concept of crowd sensing itself is built upon a large enough user basis. Therefore, it must be ensured that data recorded with the Generic App Framework cannot be used to identify individuals. With respect to this point, the most "vulnerable" aspect of the system is the app's user ID which is required to assign the recorded data to anonymous entities for aggregation purposes. Now, given that the user ID is necessary for the concept of crowd sensing to work in a wide area of applications, it must be ensured that it is generated in a way that does not allow for “reverse engineering” the user’s identity based on it.

Simply using one of the device’s hardware addresses as user ID seems risky as this approach would possibly allow for deducing the device owner’s identity by means of analyzing the device manufacturer’s supply chain and sales channels and combining the results with payment information at the point of sale. While this process admittedly involves a lot of work and is not scalable, it could still be seen as a potential weakness. Therefore, the Generic App Framework generates the user ID at its very first start based on the following pieces of information:

- The device’s MAC address
- The current timestamp with millisecond precision
- The user’s current location – including all decimal places

The individual components are then concatenated and the resulting string is hashed using the MD5 algorithm. This approach of creating the user ID can be regarded as being relatively safe as (a) the hash is irreversible and (b) even if the MAC address was known to the “attacker”, the other two components would make a brute force attack to find the correct user ID extremely time consuming.

In theory, another approach for identifying the sender of crowd sensed data could be an analysis of the IP address used for the data upload. However, since the scenarios discussed in the scope of this thesis, are all based on the use of mobile phones which do not have an IP address (only the cell phone tower used for the data transfer has one), it seems safe to neglect this attempt of identifying users.

2.5.2 Data Processing Back End

From an actual crowd sensing application point of view, the Data Processing Back End is probably one of the – if not *the* – most important components of the entire system. Its tasks are to receive, process and store the data sent out by members of the public. Consequently, the Data Processing Back End has to fulfill the following requirements:

1. It has to be scalable to ensure that it can cope with a growing user base.
2. It should be designed in a redundant way so that it can cope with partial failures of its components.
3. Both scalability and redundancy need to be achieved in a cost efficient manner as the deployment owner's budgets are often limited.
4. It needs to be extendable to ensure that it can be deployed during a wide range of crowd sensing scenarios.
5. The technologies used for its implementation should be as widely known as possible in order for it to be administrated and extended by a large developer base.

The following subsections elaborate on how the Data Processing Back End was designed in order to meet the above requirements. This work is based entirely on [30].

General Considerations

In order to meet requirements 3 and 5 from the above list, it was decided to only consider components and technologies for the design of the data recording back end, that are both free (ideally, open source) and widely used. It is the author's believe that this decision will lead to a bigger take-up of the platform than it would be the case if commercial or niche products – which might be slightly more powerful but potentially also expensive and lesser known to the developer community – were chosen instead.

At this point, it should be noted that this section does not cover details of the data visualization component as it is (a) heavily application-dependent and (b) not particularly relevant in the scope of this data-processing- and storage-focused section. Chapter 3 will present one possible data visualization technique during an evaluation of the proposed platform in a real world, large scale scenario.

Figure 2.15 presents the Data Processing Back End's general structure from an abstract point of view. It can be seen that the data sent out by smartphones is received by the back end's **Input Layer** which is essentially made up of a load balancer component. This

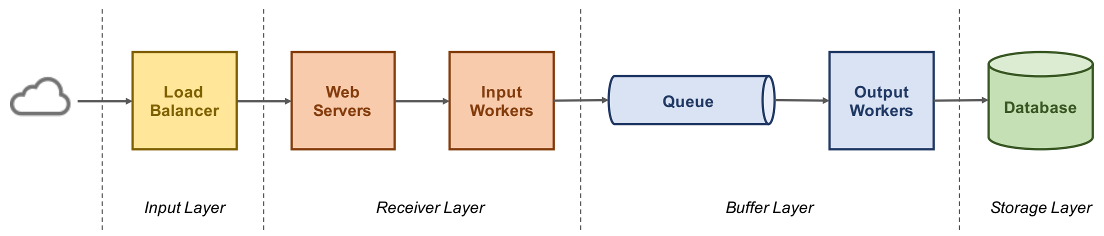


FIGURE 2.15: Overview about the general structure of the Data Processing Back End

layer presents the entry point into the back end and has the sole task of distributing the load as evenly as possible over a potentially large amount of servers. That way, the data is passed to the **Receiver Layer** which can consist of numerous identical web servers for reasons of scalability and redundancy. Furthermore, a number of input workers is managed by each web server. The web servers accept the incoming HTTP requests and forward them to the input workers which extract the data from the requests, validate it and pass it to the Buffer Layer.

Within the **Buffer Layer**, the data is being held in a queue responsible for buffering the data before it is written to the database. The main purpose of this layer is to smoothen the system's operation during peak times. To ensure scalability and redundancy, a requirement for the queue is that it can be distributed over multiple nodes. Furthermore, the queue should allow for prioritizing certain data packets which might be required by some data processing applications. An example for such an application could be the prioritization of data packets recorded during specific time frames in order to allow for generating a rough overview about the data as quickly as possible. The second component of the Buffer Layer is a host of output workers. Their task is to consume the data, process it (if necessary) and pass it to the Storage Layer. As this approach allows for data to be processed by multiple input workers, it opens up the possibility for creating multiple types of workers, each processing the data differently before storing it permanently.

Finally, the **Storage Layer** contains the actual database component. Again, this component is required to be designed for scalability and redundancy to ensure consistent read- and write-speeds as well as data integrity even in cases of partial system failures.

The following sub-sections present the design decisions made within each layer. The last sub-section then presents the packaging of all components into a set of virtual machines which can be replicated at will in order to scale the back end according to a scenario's concrete requirements.

Designing the Input Layer

The input layer's task is to receive the data from the smartphones and forward it to one of the receiver layer's web servers. This is being done with a **Load Balancer** component which is the first element of the architecture that facilitates scalability and redundancy. In general, there are three different approaches for implementing a load balancer:

- **DNS-based Round-Robin** is a solution with the least possible effort. Here, all web servers are publicly reachable with a globally unique IP address which is then assigned to a common host name. There is no further configuration required. When a client requests the host name it gets the complete list of IPs returned by the DNS resolver. Which concrete web server the client connects to is completely depending on the order of that list's entries which cannot be controlled. Hence, an even load balancing is not guaranteed. Also, redundancy cannot be achieved reliably following this approach as requests sent to a web server that is currently down, simply time out with the client receiving an error message.
- **Hardware Load Balancers** are ideal from a performance point of view. The dedicated hardware allows for maximum throughput. However, dedicated hardware can also be quite expensive – and possibly more importantly is also unflexible with respect to implementing new features.
- **Software Load Balancers** can also deliver very good performance – depending on the hardware they are running on – and are generally cheaper since they don't require specialized hardware. Furthermore, they can be extended with new functionality quite easily and also offer a whole host of additional services such as SSL offloading or VPN connections to other networks.

Since software load balancers offer nearly the same performance as hardware load balancers but come with none of their drawbacks, this approach was chosen for the implementation of the input layer. As a next step, the concrete software had to be determined.

One possible option is to use a standard web server as load balancer which is entirely possible as most available web servers offer this option via additional modules. The advantages of this approach are mostly found in the versatility of web servers: they can serve static files and perform SSL offloading directly and they can also perform prechecks of the HTTP requests immediately upon receiving them. However, process-based web servers such as the widely used *Apache HTTP Server*[3] are not ideal for being used as load balancers. The main reason for this is the fact that each incoming request is

[3]`https://httpd.apache.org`

handled in a new process which allows for great processing flexibility. On the downside, each new process requires a full copy of the memory state. The memory required for the process stays allocated until the request finishes which might take a while – especially in scenarios where the majority of the clients are connecting via a cell phone network. The consequences of this fact are large memory requirements which lead to the problem commonly referred to as *"C10k"* – the issue of one machine not being able to serve more than 10.000 clients. The alternative to process-based web servers are event-based web servers which can handle a lot more requests. One very prominent example is the *Nginx Web Server*[4]. Due to its event-based implementation, it doesn't need to keep a process alive for the entire duration of a request. On the downside, this means that it can only run non-blocking modules/logic. Within the scope of implementing a load balancer, however, this would be acceptable. For all that, the available load balancer module for Nginx is only commercially available and would therefore violate the requirement of using only free components for the proposed architecture.

The alternative to standard web server software is the use of specialized load balancer software. Here, the de-facto standard amongst the open source load balancers is *HAProxy*[5], a single-process, event-driven solution which is being shipped with most mainstream Linux distributions. HAProxy's main advantages are its light memory footprint while handling very high volumes of traffic. It also provides health-check mechanisms for connected servers and the means to control those servers – e.g. for shutting a machine down for maintenance works. However, the high performance does come at a price as HAProxy cannot decrypt HTTPS traffic which consequently needs to take place at a later stage.

Due to its high performance and the fact that within the proposed architecture, the load balancer doesn't have to perform any actual processing tasks, HAProxy was chosen for realizing the load balancer component and therefore the input layer.

Designing the Receiver Layer

Within the receiver layer, web servers collect the HTTP requests forwarded by the load balancer and check if they are valid. If that's the case, the requests are forwarded to the input workers which extract the data, validate it (i.e. check if the data has a valid JSON structure) and write it to the queue.

On this level, scalability and redundancy are achieved by employing numerous web servers in parallel. This way, incoming requests can be spread evenly over all servers – should the average load of the servers exceed a threshold, a new instance can be added easily.

[4]`https://nginx.org/en/`
[5]`http://www.haproxy.org`

Also, if one machine should fail, the load balancer will automatically take it out of the load balancing pool and continue by distributing incoming requests over the remaining machines.

The first decision that had to be made during the design of the receiver layer was regarding the type of web server. In principle, the same considerations regarding its choice as in the previous sub-section can be applied. Therefore, it was decided to use an event-based solution in order to avoid the limitations of process-based web servers which would certainly come into play in crowd sensing scenarios with a large number of clients.

In terms of event-driven web servers, the previously introduced *Nginx* and *Lighttpd*[6] have been considered. Besides their overall performance, evaluation criteria were the supported programming languages that can be used for processing the data (i.e. for implementing the input workers) and their capability to support FastCGI to forward HTTP requests to the input workers. With respect to the first two criteria, both web servers performed broadly similar. However, since Lighttpd has the capability to spawn and control FastCGI processes itself instead of having to rely on their external startup, it was chosen for the web server component over Nginx.

With respect to the input workers, the main decision to be made was regarding the programming language used for implementing them. While using C/C++ would certainly be the fastest option, the languages offer less user friendly libraries for receiving HTTP requests and for writing data to data bases. Also, the developer community able to use C/C++ is comparatively small. Programming languages supported by the web server which have much bigger developer communities are PHP and Python. Those languages do offer easy-to-use means for receiving HTTP requests and for writing data to data bases and queues. For the implementation of both the input workers, Python was chosen as it supports true object orientation, has a very clear syntax and offers more high-level data types. In addition to that, the performance of Python-workers was much superior to the one of PHP-workers (please see the following sub-section for details).

The number of input workers that are assigned to each web server can be freely configured by the system administrator via the web server's settings file during initial system benchmarks. The concrete number however, is always depending on the available memory and CPU type of the computing node. The lab system used for the implementation during the course of this work used web server nodes with 4 GB of RAM and quad-core Intel Xeon E5-2620 CPUs with a clock speed of 2.1 GHz. For this (modest) configuration, a number of 10 input workers per web server proofed to be a good value for handling scenarios with more than 10'000 concurrent users.

[6]`https://www.lighttpd.net`

Designing the Buffer Layer

The buffer layer's main task is to decouple the receiver layer from the storage layer in order to handle potential imbalances between read- and write-performances during peak times. It therefore holds received data objects in a queue component which is fed by the input workers from the receiver layer. The buffer layer's output workers are then consuming the data and write it into the data base.

In general, there are two approaches to designing the queue component:

1. Using **one single queue** running on a separate node would allow for optimal load balancing in terms of read-performance as all output workers (which are potentially running on different nodes) have access to all of the data and can therefore always be kept busy. The big downside however is that a failure of the queue would bring down the entire system and might also result in a complete loss of the data that was held in the queue at the time of the crash.
2. Having **one queue per web server** on the other hand, does not allow for load balancing of the read operations on a global level since the output workers are only connected to one queue and can therefore only consume the data that is being written to that particular queue. Hence, if one queue is empty, the connected workers must also pause, regardless of the state of the other queues. However, the big advantage of this approach is the fact that the failure of one queue does not result in a complete system failure.

It was decided to design the buffer layer according to approach number 2 in order to fulfill the overall system's requirement of redundancy. Also, the disadvantage in terms of load balancing read operations on a global level seems to be negligible as the input layer already achieves an even distribution of the incoming data over the web servers. It therefore stands to reason that based on identical node hardware, the web servers will perform broadly on the same level which will lead to the queues being filled with more or less the same amount of data. Furthermore, this approach also facilitates horizontal scalability with respect to dealing with increasing amounts of incoming data.

For the actual implementation of the queue, the two popular open source queue solutions *ZeroMQ*[7] and *Beanstalkd*[8] were considered. While both offer a similar feature set, it was decided to use Beanstalkd as it has been optimized for running time-consuming tasks asynchronously which is not true for the more generic ZeroMQ. Furthermore, Beanstalkd

[7]`http://zeromq.org`
[8]`http://kr.github.io/beanstalkd/`

offers two features that are not implemented in ZeroMQ: it supports prioritization of certain data objects and binary logs. The latter is especially relevant in case the application (or the node itself) is crashing. After a reboot, the data that was still in the queue at the time of the crash is read from the disk and re-inserted into the queue. Hence, unless a node's disk gets destroyed, it is more or less guaranteed that data arrives in the storage layer, eventually.

Beanstalkd is implemented as a job queue meaning that all data objects are represented as jobs which can have a certain status. After a job gets inserted into the queue, its status is set to `READY` – any connected output worker may now process the job. If a worker sets a job's status to `RESERVED`, it means that all the other workers do not have access to it anymore. Once a job has been completed (i.e. the data has been passed on to the storage layer by an output worker), it is being deleted from the queue.

The considerations regarding the number of output workers assigned to each queue are the same as the ones for the input workers discussed in the previous sub-section. Within the lab-system the number was set again to 10 in order to match the number of input workers.

The output workers have also been implemented using the Python programming language for the same basic reasons as mentioned above in case of the input workers. Once both types of workers have been implemented, a load test (using the load testing tool *FunkLoad*[9]) was performed in order to evaluate if the Python or the PHP implementation performs better. With 500 concurrent users sending data to the back end, the PHP implementation had an average request response time of about 0.067 seconds. The Python implementation on the other was much quicker: even with 7'000 concurrent users it managed to achieve an average response time of about 0.03 seconds. Consequently, Python was chosen for the final implementation of the workers.

Designing the Storage Layer

The storage layer consists only of a data base (DB) component and must fulfill the following requirements:

1. **Scalability**: multiple DB nodes have to be supported
2. **Reliability**: the failure of a single DB node must not take down the whole system
3. **Distributed Write Operations**: in order to also scale the DB's write-performance, it must be possible to write to several nodes simultaneously

[9]`https://funkload.nuxeo.org`

For traditional SQL data bases (e.g. *MySQL* or *PostgreSQL*), scalability is a challenge on its own – especially, when it comes to horizontal scalability [31]. The requirement for supporting distributed write operations however, excludes the entire category of free SQL data bases from the list of possible choices for implementing the Data Processing Back End's storage layer. The reason for this is that SQL DBs can only mirror data from one master to multiple slaves in order to allow for a distribution of read operations – the concept of parallel writes to multiple replication masters cannot be realized due to the absence of a global locking concept. In practice, this would mean that two clients were able to update the same data row on different nodes with different values.

As a consequence, the category of No-SQL data bases needs to be considered. Within this concept, there's no schema that data needs to adhere to. Instead, data is organized in form of separate documents belonging to a collection. Therefore, from a conceptual point of view the data structure handling is moved from the storage layer to the application layer.

At the point of the system design, the two most popular No-SQL data bases were *CouchDB* [10] and *MongoDB* [11]. During the evaluation, it was decided relatively early on, that MongoDB would be the tool of choice for implementing the storage layer. The first main reason was that it is designed from ground up to be scalable. Secondly, its binary protocol is a lot faster than CouchDB's HTTP protocol. Here, a lot of overhead is introduced due to the fact that more data has to be transferred for each request. This is especially true for authentication, which takes place during each request. In scenarios with a large amount of write operations (as it is the case in crowd sensing applications), this leads to a significant disadvantage compared to MongoDB's efficient binary protocol.

Within MongoDB, both the data and the queries are encoded as BSON files by the client library (binary JSON) for maximum speed. Also, the DB's object IDs have been designed for multi-node deployments: they consist of 12 Bytes and have the format `[timestamp, host name, PID, increment]` – consequently, there's no need for a global lock to find the correct object ID as the ID itself already contains the host where the data is stored. During insert operations, the DB only checks if an object ID is present and if the size limit of 4 MB hasn't been exceeded. This is both an advantage from a performance point of view but also has positive aspects from a security perspective as the possibility for injection attacks during insert operations is being eliminated. Another positive aspect of MongoDB from a crowd sensing perspective is the support for batch-inserts where multiple inserts into the same collection are performed with just one TCP request – an ideal way to insert sensor data where it's usually the case that the contents of one data

[10] `http://couchdb.apache.org`
[11] `https://www.mongodb.com`

packet are saved to the same collection. Finally, MongoDB supports geospatial queries which can be a meaningful benefit for some crowd sensing applications dealing with location data. Using such queries allows for example to efficiently retrieve all objects within a given distance to a geographic location.

In order to realize scalability of read operations and redundancy, the concept of MongoDB's *replica sets* has been used. It allows for replicating data from a master node to one or more slave nodes. Each replica set must be assigned a unique name and may only contain one master node. The replication of data from the master node to the slave nodes is being done automatically. If a new slave node should be added to the replica set, it only needs to know the name of the set and the address of one of the set's members; the integration of the node into the replica set is again handled automatically. If the master node should fail, its role is transferred to one of the slave nodes.

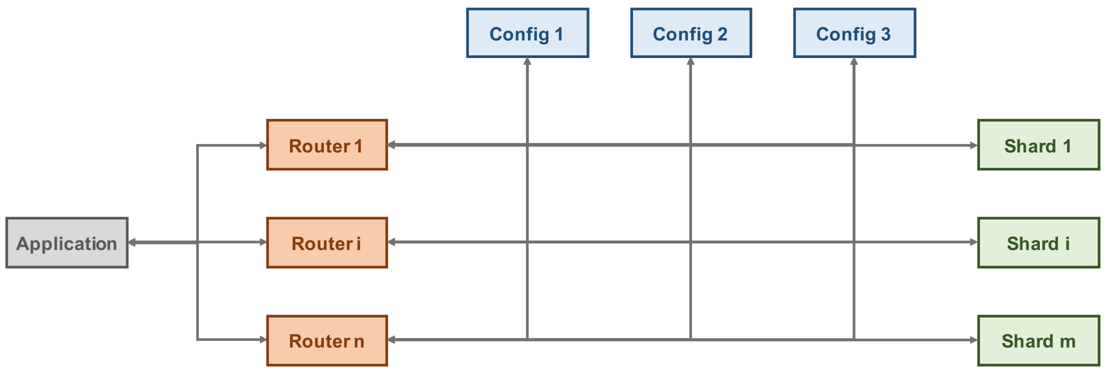


Figure 2.16: Layout of a MongoDB sharding cluster

Realizing the scalability of write operations requires MongoDB's concept of *sharding*. This concept is similar to partitioning a relational data base – however, MongoDB mostly automates this process. Figure 2.16 shows the layout of a so-called "MongoDB Sharding Cluster". Within such a cluster, a collection is split up into so-called "chunks", i.e. portions of data that are created based on a "sharding key". These chunks are stored on "shards", whereas a shard can either be a single MongoDB process or – if redundancy is required in a production environment – a replica set.

An application (e.g. an input worker) accessing the MongoDB sharding cluster however does not need to be aware of the "chunking-logic" or the sharding key. This task is handled by a MongoDB Router which is accepting all incoming connections and is placed in front of all shards. To ensure reliability, routers can also be scaled over multiple nodes. Furthermore, a new component is introduced in the scope of a sharding cluster: the "MongoDB Config Server" which has to exist in exactly three instances in a production environment. Its task is to save and control the cluster configuration and it holds information about whether or not a data base or a collection is sharded, about the definition of the sharding key and also about which shards are defined and where they are stored. In a sharding cluster, each config server has to be connected to each router and to each shard. Furthermore, each router has to be connected to each shard.

One of the most important aspects for the performance of a sharding cluster is the right choice of the sharding key. Generally speaking, sharding keys with a low cardinality are not suitable – e.g. choosing the day of the week as sharding key would result in only 7 possible shards. Ideal sharding keys are designed as composite keys: for example, the combination of the year and month of the data plus its user ID would result in a potentially large number of chunks which can be distributed over many shards – the precondition for scaling a data set efficiently.

Packaging the Data Processing Back End

The previous sub-sections have introduced the overall architecture of the Data Processing Back End and the implementation details of each component. Now, the last step is to assign those components to a set of virtual machines (VMs), in order to be able to create scalable deployments. Therefore, Fig. 2.17 presents the proposed VM-layout. Please note that for reasons of clarity, the detailed interconnections between the different MongoDB sharding cluster components have been simplified in this figure.

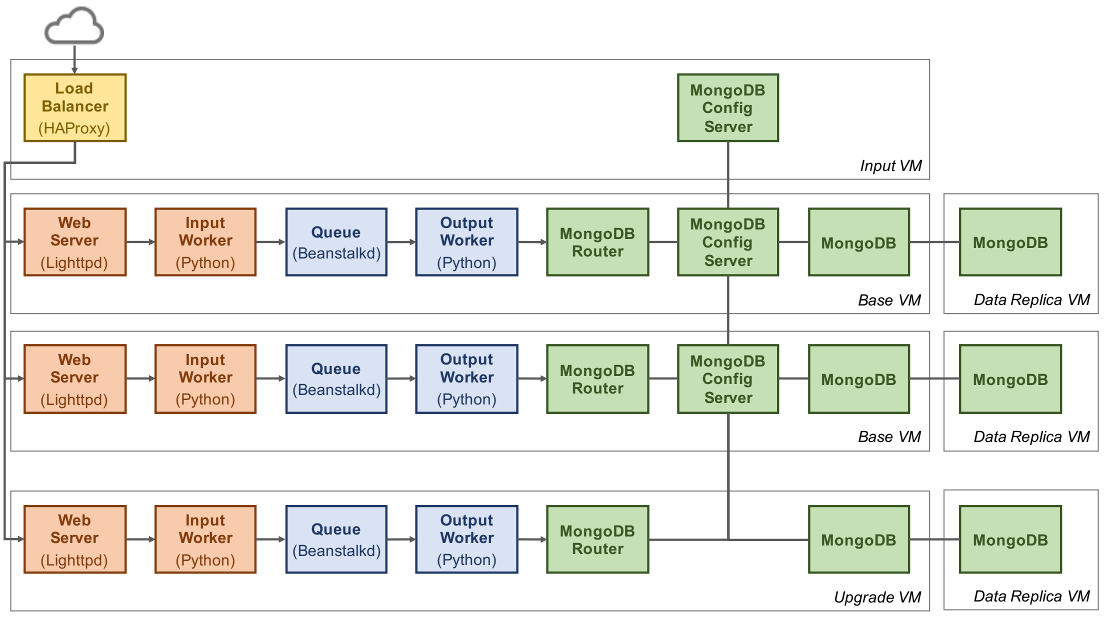


FIGURE 2.17: VM-layout of the Data Processing Back End architecture

A minimal configuration requires exactly one Input VM and two Base VMs. If basic redundancy should be achieved, one Data Replication VM must be added to each Base VM in order to be able to create a minimal MongoDB replica set consisting of two MongoDB storage instances. If any of those VMs should fail, it must be replaced with an identical VM. Using the aforementioned computing nodes with 4 GB of RAM and quad-core Intel Xeon E5-2620 CPUs with a clock speed of 2.1 GHz for each VM in a lab environment, the resulting configuration was easily able to handle a test scenario with 10'000 concurrent users sending data to the back end (see next sub-section for details).

If the deployment scenario's size calls for it or if a higher degree of redundancy needs to be achieved, the back end's capacity can be scaled up by adding any number of upgrade VMs with at least one Data Replica VM each. In the interest of more robust data redundancy however, it is advisable to add at least two Data Replica VMs to each Base or Upgrade VM. That way, the resulting MongoDB replica set consists of at least three storage instances and can therefore still operate reliably with remaining basic redundancy in case one of those instances goes down.

Benchmarking the Data Processing Back End

In order to find out how much load a minimal configuration of the Data Processing Back End is able to handle, a stress test using the aforementioned FunkLoad load testing tool was performed. The tool was configured to simulate the behavior of real crowd sensing apps which typically upload data packets to the back end in one minute intervals – for the benchmark, each data packet contained 60 GPS measurements, consisting of the location itself, speed and heading values, and a precision value. In total, four test runs were performed with each simulating 1'500, 3'500, 7'500 and 10'000 concurrent users, respectively. Within a benchmark run, the simulated users would send their upload requests evenly distributed over the period of one minute. Figure 2.18 shows the results of these benchmarks.

The figure plots the number of concurrent users against the achieved request response times – i.e. the time it took until an uploaded data packet was stored safely in the system. It can be seen that even for 10'000 concurrent users, the slowest request response time was less than 0.055 seconds with the fastest being a bit more than 0.015 seconds. Furthermore, 90 % of the uploads were stored between 0.027 and 0.044 seconds, with the fastest 10 % of the uploads even being processed within 0.019 to 0.027 seconds.

It can also be seen that the overall difference in the system's performance doesn't vary greatly between 1'500 concurrent users and 10'000 concurrent users. This observation suggests that the Data Processing Back End could deal with more load even in its minimal configuration. However, due to the limitations of computing resources in the

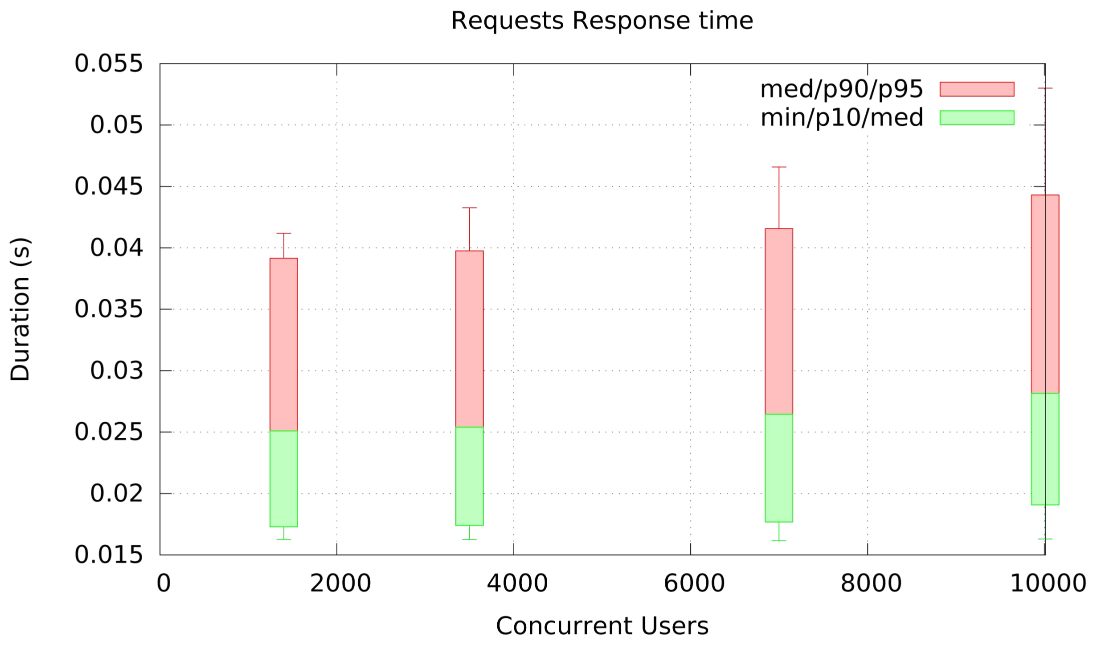


FIGURE 2.18: Results of the Data Processing Back End benchmark

lab environment, it was not possible to perform benchmarks with a larger number of concurrent users – the 10'000 user simulation already required 14 computing nodes for creating the necessary amount of upload requests with the FunkLoad tool. In summary, it can be said that the Data Processing Back End manages to achieve a satisfactory performance.

2.5.3 Content Creation Back End

The Content Creation Back End has the important task of keeping the crowd sensing apps up to date. This applies to their feature set, design, sensing behavior and app module contents (e.g. map information, messages, etc.). Given the fact that all app instances are polling the back end in regular intervals with update requests (and assuming that crowd sensing is not performed on a 24/7 basis), it is also the architecture's component that has to deal with the highest average load. It therefore has to meet the following requirements:

1. Scalability is required in order to ensure that all deployed apps are provided with updates reliably, regardless of the number of app users.
2. The back end has to be designed in a redundant way to ensure that partial system failures don't result in a service down time.
3. The system has to be designed with minimal running costs in mind given the facts that large numbers of users are expected and deployment owners' budgets are often limited.

In the following sub-sections, the Content Creation Back End's design is presented. Also, measurements that have been taken in order to meet the above requirements are elaborated.

General Considerations

Figure 2.19 shows the structure of the Content Creation Back End. In order to meet requirement number 3 from the above list, the system has been designed in a way that it can be easily deployed on the infrastructure of a commercial cloud computing provider, which often has cost advantages compared to a dedicated self-managed server. In fact, components with a striped background are dedicated cloud services – the other components can of course also be hosted internally, in case this is preferred by the deployment owner.

At this point, it needs the be mentioned that the Data Processing Back End could also be deployed in the cloud. However, from a data protection point of view, that system is much more sensitive than the Content Creation Back End and the deployment owners are therefore likely to prefer it running on internal hardware in order to have full ownership of – and control over – the recorded data.

The **Content Creation Layer** is exclusively used by the deployment owner's staff. For reasons of redundancy, it contains a number of computing nodes (accessed via a load

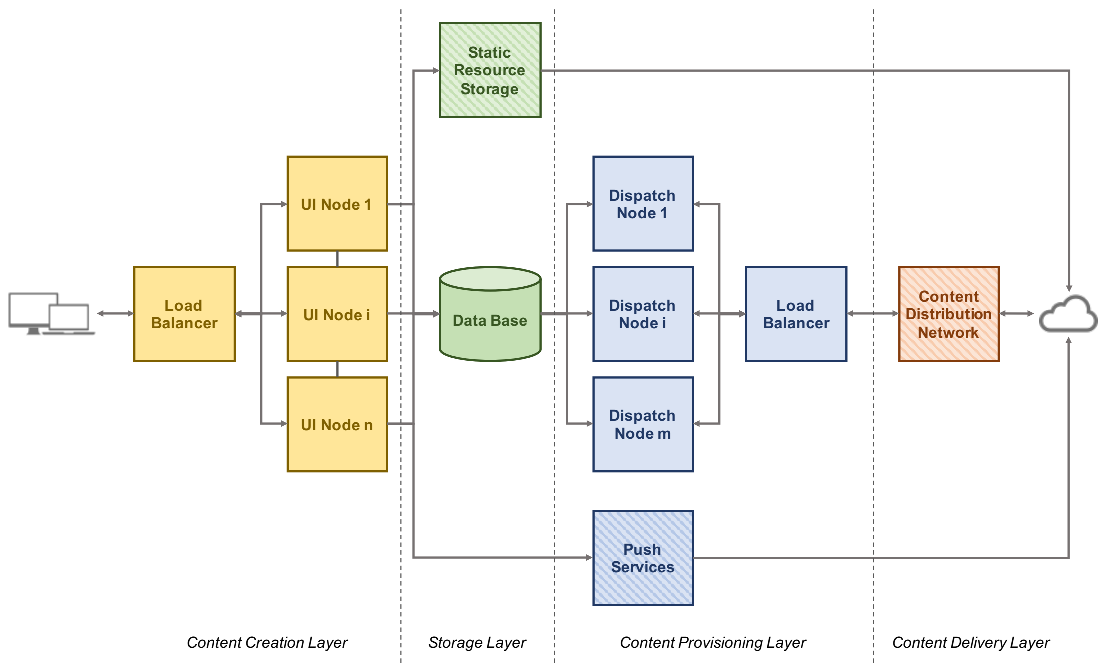


FIGURE 2.19: Overview about the structure of the Content Creation Back End

balancer) providing the system's user interface (UI). This UI has already been described in section 2.4 and is used to administrate the system and create the app resources. Those resources are stored within the **Storage Layer** which consists of two individual components: a local data base and a cloud-based storage. While the former holds the data required for building the app resource JSON files, the latter contains static files such as images and icons.

The **Content Provisioning Layer** is responsible for actually building the app resources and for making them available to the apps. This task is taken over by a number of dispatch nodes which are made available to access from the internet via a load balancer. The third type of component within this layer are the push services. Those services are hosted by the smartphone operating system vendors Google and Apple and are required for delivering push notifications to apps. The UI application accesses those services directly whenever there is a need for one of the modules to send a push notification – e.g. in case of sending important messages to app users.

The final layer is the **Content Delivery Layer** which basically acts as a buffer to reduce the load on the Content Provisioning Layer. This is realized by using a cloud-based content delivery network which buffers replies to known requests and delivers them immediately without passing the request on to the Content Provisioning Layer. The logic behind this is that the majority of update requests sent from the apps can either be replied with sending an HTTP status 304 (i.e. *"no update available"*) or with a reply that has already been sent out previously (e.g. the latest map data). By lowering the load on the Content Provisioning Layer, operating costs can be reduced dramatically, as the majority of the app requests are handled by the content delivery network which is comparably cheap.

Content Creation Layer

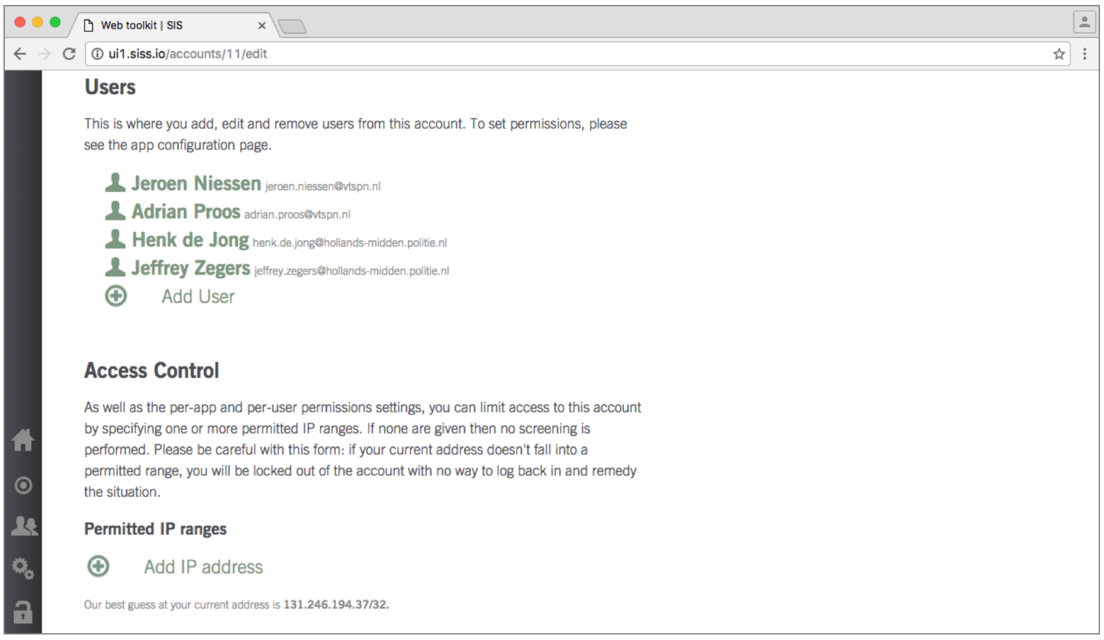
The Content Creation Layer has the main task of running the user interface (UI) in a reliable fashion. Given that the number of users interacting with the UI is expected to be comparatively small, scalability is no issue. However, in order to provide reliable access to the UI in case of partial system failures, redundancy has been built into this layer.

Therefore, users working with the UI are accessing it via a load balancer managing a number of computing nodes running the UI application. For basic failure tolerance, at least two UI Nodes should be deployed – however, a larger number is advisable for increased resilience. When running the Content Creation Back End in a cloud environment, the cloud computing operator usually offers specific load balancers which can be used for this application without a problem. This approach has the advantage of being able to utilize other services based on that load balancer's capabilities – a relevant example would be automatic spawning of a new UI Node in case one of them crashes. In case the back end is deployed on a self-managed server, the load balancer component is realized using HAProxy, which has been introduced in the previous section.

The UI Nodes are solely responsible for running the UI application which has been implemented as a lightweight Ruby on Rails application utilizing HTML5 and JavaScript for the front end. Those technologies have been chosen because they are widely known within the developer community which makes it easy to add new modules to the UI application. Since the features of the UI were already presented in section 2.4, this topic will not be elaborated at this point. The settings made via the UI and the contents created with it are kept within the Storage Layer which is discussed in the scope of the next sub-section.

Given the fact that multiple users might be working within the same account on the same app while connected to different UI Nodes, it must be ensured that the different UI sessions are always synchronized. This is accomplished by means of a shared pub/sub-channel where all the relevant changes made via the UI are published.

An important aspect of the Content Creation Layer is security. If an unauthorized person gains access to the UI, the consequences could theoretically be severe. The very first measure to manage that risk is of course the fact that people need to authenticate themselves with a user name and a password. The UI also enforces a strict password policy which includes the need to change the password on a monthly basis. Another layer of security is the role-based access to the UI which has already been presented, previously. By establishing that not every user has access to all the system features, it is ensured that stolen account credentials can always be used to access all UI features.



Users

This is where you add, edit and remove users from this account. To set permissions, please see the app configuration page.

- Jeroen Niessen jeroen.niessen@vtspn.nl
- Adrian Proos adrian.proos@vtspn.nl
- Henk de Jong henk.de.jong@hollands-midden.politie.nl
- Jeffrey Zegers jeffrey.zegers@hollands-midden.politie.nl
- Add User

Access Control

As well as the per-app and per-user permissions settings, you can limit access to this account by specifying one or more permitted IP ranges. If none are given then no screening is performed. Please be careful with this form: if your current address doesn't fall into a permitted range, you will be locked out of the account with no way to log back in and remedy the situation.

Permitted IP ranges

Add IP address

Our best guess at your current address is **131.246.194.37/32.**

FIGURE 2.20: Screenshot of the UI's access control screen

As a "last line of defense", the UI's access control screen (see Fig. 2.20 for reference) provides an IP filter. If this option is enabled, UI access can be restricted to a certain IP range. While such a measure means that the UI can't be used outside a set of well defined machines, it does severely hamper the ability of unauthorized persons to access the UI with stolen credentials.

Storage Layer

The sole task of the Storage Layer is to reliably store all data required for building the app resource files and for running the UI (e.g. user data, etc.). For reasons of cost-efficiency, this layer has been divided into two components.

The first component is the Data Base which holds the data for running the UI application. In addition to that, it contains the attributes and settings for building the app resource files. Listing 2.7 once again shows an excerpt of such an app resource file. The entire contents of the JSON are considered as attributes and are therefore held in the Data Base – this also includes the URLs to static resources as it can be seen in line 4 of the listing.

```
// <...>
{
  "title": "Map",
  "iconpath": "http://[url]/TabBarIcon1-Maps.png",
  "directcontroller": "MapsViewController",
  "startstring": "" },
// <...>
```

Listing 2.7: Excerpt of an app resource file

The static resources themselves however, are stored in the cloud-based Static Resource Storage where they can be accessed directly by the apps via the link given in the JSON structure. The decision to store static files on a cloud-infrastructure was made based on cost and performance considerations. From a quality point of view, it is very important for the apps to be able to download static resources as quickly as possible as users may otherwise experience unnecessary waiting times (e.g. when opening a message immediately after it arrived in the app and having to wait for the image to load). On the other hand, delivering image files to a large number of users at maximum speed usually requires a lot of effort on the server-side and a substantial bandwidth. Both these facts are directly connected to high costs.

When storing image files on a cloud-based infrastructure however, the necessary performance requirements are met since the cloud providers usually cater to much more demanding needs (e.g. Amazon Web Services hosting the video-on-demand provider Netflix). From a cost perspective, cloud storage offerings are also very attractive: at the time of the writing of this thesis, Amazon Web Services' EC2 storage service charged only 0.023 US$ per GB per month for storing data and 0.09 US$ per GB of outgoing traffic (with the first GB being free of charge each month) – on top of that, a request-based fee of 0.004 US$ per 10'000 HTTP GET requests is being charged [12]. Taking those numbers into account, it becomes obvious that for the scale of most crowd sensing applications, it makes sense to host static app resources with a cloud storage provider.

From an implementation point of view, the Data Base is not very demanding. The load with respect to write operations is fairly limited since data only needs to be written when the settings or contents of an app are updated. The same is true for read operations which take place either after content updates or after UI settings have been changed. Also, the number of clients performing those read operations is limited to the UI Nodes and Dispatcher Nodes. Therefore, scalability is not of utmost importance within the Storage Layer.

[12]For pricing details please refer to `https://aws.amazon.com/s3/pricing`

When looking at the structured nature and the volume of the stored data, it becomes obvious that a normal SQL-data base is absolutely suitable for being used as the Data Base component. In the proposed system, *MySQL*[13] was chosen – purely based on the fact that it is extremely popular in the developer community and a lot of developers have experience using it. With respect to redundancy, MySQL also supports master-slave replication to ensure that the data is still accessible when one DB node goes down.

Content Provisioning Layer

The Content Provisioning Layer's main responsibility is to build the app resource files based on the Data Base's contents and to deliver them to the apps. This task is accomplished by a set of Dispatcher Nodes which is made available to the internet by a load balancer.

Furthermore, the Content Provisioning Layer contains the Push Services component, which is again hosted outside the actual system. Some app modules might require push messages when new contents have been created – an example would be the messaging module which can send push notifications alerting the app users to the presence of a new message in case that message is so important that the deployment owner wants all app users to be aware of it as quickly as possible. As push notifications are a service that's managed by a smartphone's operating system, they are delivered via a service owned and hosted by that operating system's vendor (i.e. Google or Apple). If a module creates a piece of content that requires a push notification to be sent to the apps, the UI Nodes directly trigger that push notification after they have written all the required information to the Data Base. This process takes place by supplying the Push Services with the notification contents and a list containing the device identifiers of all devices that should receive the push notification.

The Dispatch Nodes are again implemented as a lightweight Ruby on Rails application. Whenever the UI Nodes performed an update to one of the app resource file's attributes, they trigger the Dispatch Nodes to also update themselves. This takes place by polling the Data Base for the required attributes and building the latest version of that respective app resource. At this stage, an ETag (see section 2.5.1) is also associated with the new resource. For the sake of efficiency, the Dispatch Nodes keep all app resources in memory – given the small size of those resources however, this does not impact hardware requirements in a meaningful way (a node with 8 GB of RAM is sufficient). For reasons of scalability and redundancy, there are multiple Dispatch Nodes available – while the exact number does of course depend on the deployment scenario's size and character, there was never a need for more than three nodes in the course of the writing of this

[13]`https://www.mysql.com`

thesis. As mentioned earlier, those nodes are made available to the apps via a load balancer. Again, this load balancer can either be implemented using HAProxy or by utilizing a cloud provider's solution.

When an app is sending an update request to the Content Creation Back End (again, see section 2.5.1 for reference), that request is eventually processed by a Dispatch Node. First, the dispatch application compares the ETag supplied with the request to the requested resource's current ETag – if the two match, it responds immediately with an HTTP status code 304 (i.e. *"no update available"*). If the ETag provided with the request is older than the current one, the dispatch application returns a response containing the latest version of the requested app resource.

Content Delivery Layer

In the previous sub-section it was mentioned that scalability of the Content Provisioning Layer is achieved by running a multitude of Dispatch Nodes and that within the scope of this work, there was never a need to employ more than three nodes from a performance point of view. However, this number is only achievable due to the presence of the Content Delivery Layer.

If one assumes that a large deployment of the system may include 50'000 app installations with 10 active modules contacting the back end with update requests once every minute (a realistic value in some deployment contexts), this leads to a total of 500'000 update requests per minute – or approximately 8'333 update requests per second. Processing this amount of traffic (and responding to it) directly within the Content Provisioning Layer would require a larger amount of Dispatch Nodes which in turn would lead to increased running costs of the system.

The concept of the Content Delivery Layer is based on the fact that the cardinality of the possible responses to the update requests is rather small: responses either only contain the HTTP status code 304, the latest version of the requested app resource or – in case of app resources that are updated incrementally – the delta between the current version of the resource and the version on the requesting device. Depending on the number of app content updates that are performed by the deployment owners, the set of possible responses to update requests may only include a few hundred elements, if at all.

Due to these circumstances and in combination with the fact that app resources are simple JSON files, the problem of supplying apps with up to date contents has been reduced to the basic problem of making a limited amount of static resources available at the highest possible speed to a client. This exact problem has been solved by content delivery networks (CDNs) – which were originally developed for delivering large media files efficiently – for a long time.

It was therefore decided to connect the Content Provisioning Layer's load balancing component to a CDN. That CDN has then been configured to use the combination of a request's URL and the ETag within the request's HTTP Header as key for its buffering strategy. Initially, all request/ETag combinations are unknown to the CDN, so it passes them on to the Content Provisioning Layer. However, with each reply from that layer, the CDN "learns" about a new response and buffers it internally. Furthermore, the CDN is configured to flush that buffer every x seconds. This flushing mechanism is necessary in order for content updates to actually make it to the CDN. Without it, requests actually requiring a new response (because the associated resource has been updated), would receive the same old buffered response over and over again. Thanks to the flushing mechanism however, the requests do get through to the Dispatch Node eventually, which gives the dispatch application the chance to pass new contents into the CDN.

The choice of x is of course crucial for the overall performance of the system. If it is set too high, there is a real chance that updates don't get delivered to the app within the responsible update channel's update frequency which would lead to a decreased quality of service. Of course, there's always the mathematical chance that an update call misses a new resource due to the buffering strategy but the lower the value x is set, the smaller that chance. Within the scope of this work, x was set to 3 seconds. In practice this means, that the Dispatch Nodes are hit with each type of request on average once every 3 seconds – all other requests of that type are caught by the CDN.

If the CDN provider is chosen smartly according to the application scenario, the CDN component is also extremely cost efficient. The application at hand is characterized by a large number of request and a comparatively very small data volume per response – even the largest app resources are hardly more than a few dozen KB in size. It is therefore smart to chose a CDN provider which charges based on traffic volume. During the course of this work, *MaxCDN*[14] has been chosen as CDN provider. Within their North American and European pricing region, MaxCDN charge 0.06 US$ per GB of traffic[15], which makes the approach economically much more feasible than investing in a larger number of Dispatch Nodes.

[14]`https://www.maxcdn.com`

[15]For detailed pricing information, please see `https://www.maxcdn.com/pricing/`

Benchmarking the Content Creation Back End

In order to also find out how the Content Creation Back End performs under heavy user load, a benchmark with just one Dispatch Node was performed using the *Blitz* performance testing tool[16]. During that test, the back end was hosted in an Amazon Web Services EC2 environment, with all computing nodes running on small instances. At the point of the benchmark, the EC2 website stated that a small instance *"is roughly comparable to single-core 1.7GHz CPU machines with 1.7GB of RAM"* – hence, the computing performance was very limited. Figure 2.21 shows the results of this test.

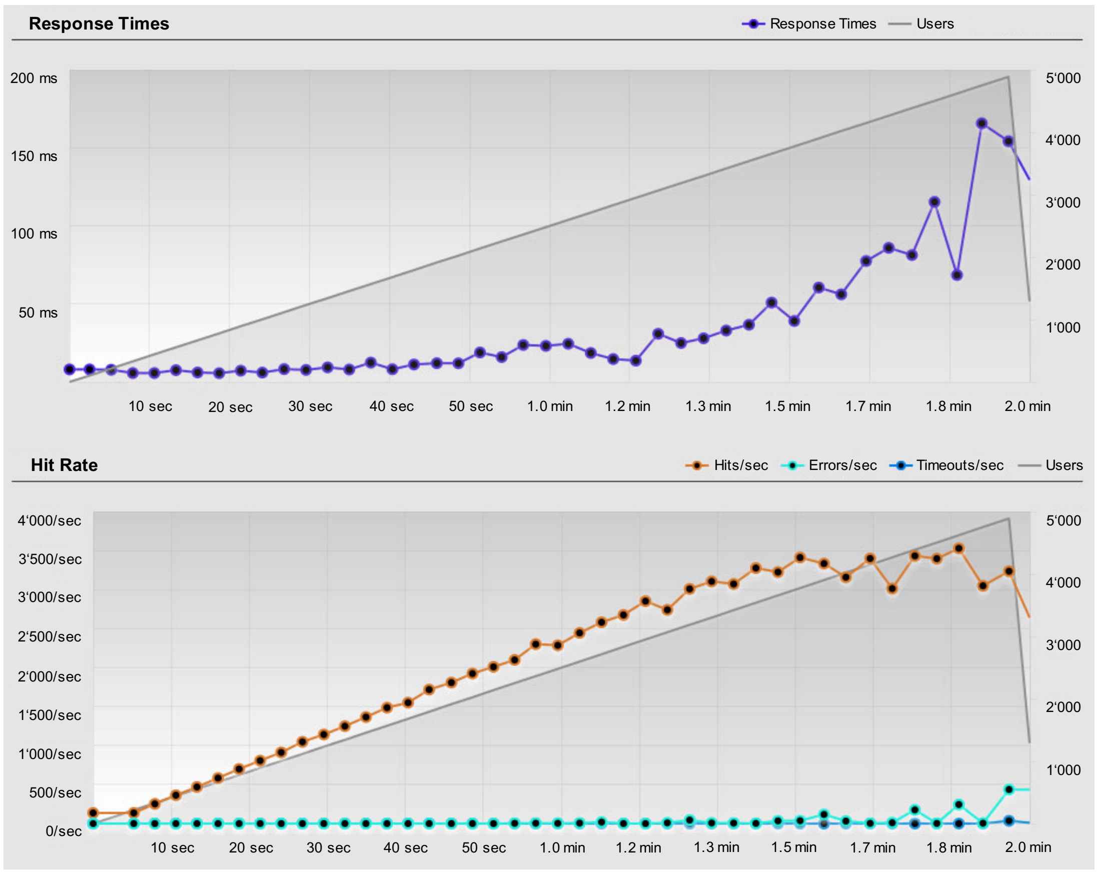


FIGURE 2.21: Results of the Content Creation Back End benchmark

The test was performed over a duration of 2 minutes – during that time, the number of apps performing simultaneous update requests was increased to 5'000 in a linear fashion. The lower graph shows the hit rate (i.e. the number of requests per second), the error rate and the timeout rate over time. The upper graph shows how the corresponding request response time developed over the course of the test.

It can be seen that the configuration with a single Dispatch Node manages to deal with a total load of around 3'500 requests per second before the hit rate bottoms out and errors

[16]`https://www.blitz.io`

are beginning to show up. At approximately the same load, the response time starts to increase significantly. It can therefore be concluded that a minimal dispatch configuration of the Content Creation Back End is able to deal with 3'500 update requests per second. Consequently, in a scenario where an app has ten modules performing update calls once every minute, this configuration could serve about 21'000 app users per Dispatch Node. On more capable hardware, this number can be expected to be considerably larger.

It is important however, to note that this performance was achieved with the Content Delivery Layer in place. In order to find out how much that layer contributes to the overall performance, another test without it was performed. Consequently all the requests had to be handled by the single Dispatch Node directly. During this test, the system's performance bottomed out at approximately 50 requests per second – or roughly 300 app users. This test clearly shows the value of the Content Delivery Layer.

2.5.4 Communication Logic

After the architecture of the proposed Data Collection Platform's components has been elaborated in detail, this section presents the most important communication processes between those components. Figure 2.22 illustrates how the Content Delivery Network "learns" to respond to update requests by buffering responses from the Dispatch Nodes. This way, the CDN can take a large amount of load from the actual Content Creation Back End and therefore helps with improving both the system's overall-performance and cost effectiveness. In this example, the update requests from the app are responded to with HTTP status 304 – i.e. there are no updates to the requested app resource available.

Figure 2.23 on the other hand shows how the process of updating app resources takes place when an app really is out of date. In this example, the CDN also has the required response buffered so that the Dispatch Nodes don't even have to be contacted in order to deliver the updated resource to the app. At the end of this update process (step 13), all app modules that are registered as listeners for that particular resource are notified of the update and can react accordingly.

One example for such a module is the app's Sensing Manager which is responsible for recording the crowd sensing data and is registered as listener for the sensing information. This resource contains information about when and where, which recording should be performed. Fetching the updated sensing information is also part of Fig. 2.24 (step 1) which illustrates the process of recording and uploading crowd sensing data to the Data Processing Back End.

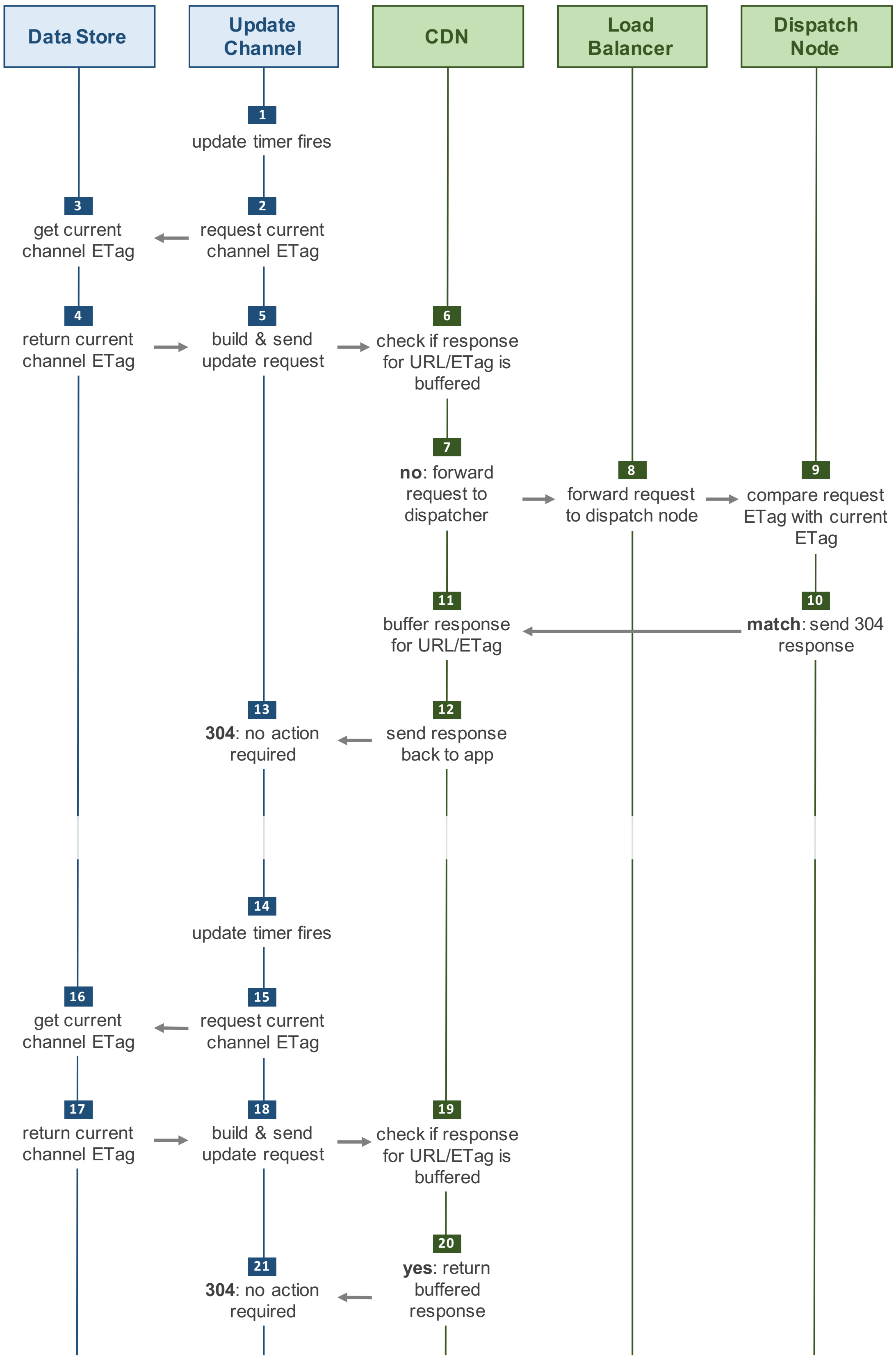

Data Store
Update Channel
CDN
Load Balancer
Dispatch Node
1 update timer fires
2 request current channel ETag
3 get current channel ETag
4 return current channel ETag
5 build & send update request
6 check if response for URL/ETag is buffered
7 **no**: forward request to dispatcher
8 forward request to dispatch node
9 compare request ETag with current ETag
10 **match**: send 304 response
11 buffer response for URL/ETag
12 send response back to app
13 **304**: no action required
14 update timer fires
15 request current channel ETag
16 get current channel ETag
17 return current channel ETag
18 build & send update request
19 check if response for URL/ETag is buffered
20 **yes**: return buffered response
21 **304**: no action required

FIGURE 2.22: CDN buffering process: the blue entities are part of the Generic App Framework, the green entities are part of the Content Creation Back End.

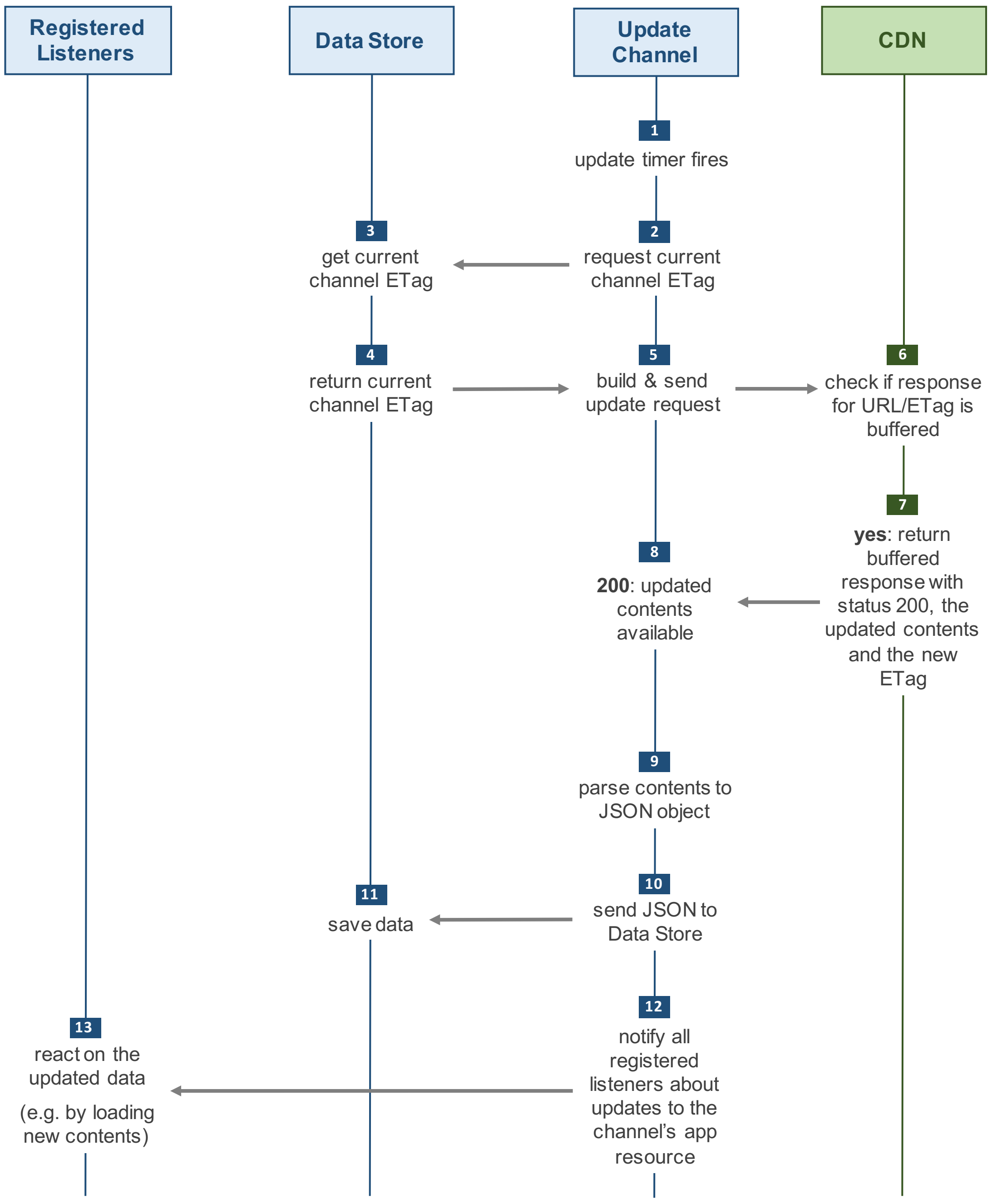

Registered Listeners
Data Store
Update Channel
CDN
1 update timer fires
2 request current channel ETag
3 get current channel ETag
4 return current channel ETag
5 build & send update request
6 check if response for URL/ETag is buffered
7 yes: return buffered response with status 200, the updated contents and the new ETag
8 200: updated contents available
9 parse contents to JSON object
10 send JSON to Data Store
11 save data
12 notify all registered listeners about updates to the channel's app resource
13 react on the updated data
(e.g. by loading new contents)

FIGURE 2.23: App resource update process: the blue entities are part of the Generic App Framework, the green entity is part of the Content Creation Back End.

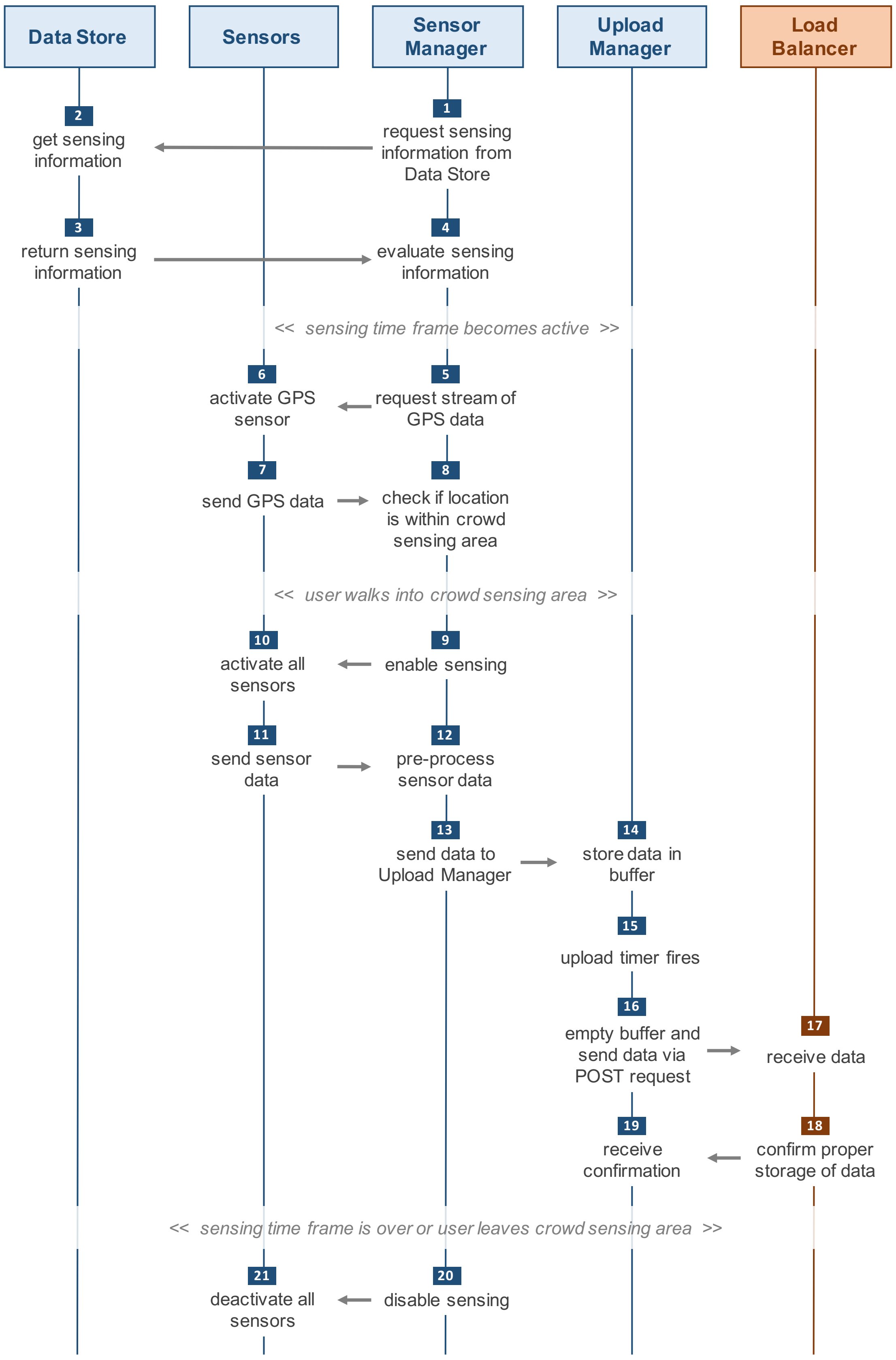

Data Store
Sensors
Sensor Manager
Upload Manager
Load Balancer
1 request sensing information from Data Store
2 get sensing information
3 return sensing information
4 evaluate sensing information
<< sensing time frame becomes active >>
5 request stream of GPS data
6 activate GPS sensor
7 send GPS data
8 check if location is within crowd sensing area
<< user walks into crowd sensing area >>
9 enable sensing
10 activate all sensors
11 send sensor data
12 pre-process sensor data
13 send data to Upload Manager
14 store data in buffer
15 upload timer fires
16 empty buffer and send data via POST request
17 receive data
18 confirm proper storage of data
19 receive confirmation
<< sensing time frame is over or user leaves crowd sensing area >>
20 disable sensing
21 deactivate all sensors

FIGURE 2.24: Data recording process: the blue entities are part of the Generic App Framework, the dark orange entity is part of the Data Processing Back End.

2.6 Chapter Summary

This chapter presented the proposed Data Collection Platform consisting of a Generic App Framework for efficiently building crowd sensing smartphone apps, a Content Creation Back End for configuring those apps and for providing them with contents, and a Data Processing Back End for receiving, processing and storing the crowd sensing data recorded by the apps. After an overview was given about how the system is being deployed and used, the architecture and implementation details of each component was elaborated.

The introductory section stated a number of research questions to be answered within the scope of this chapter. In the following, this work's main contributions with respect to each of those questions are summarized again.

How can crowd sensing applications be efficiently deployed at scale?

It was shown in detail how effortlessly a crowd sensing application can be deployed with the help of the platform's cloud-based web interface. This tool allows for building, designing and managing native crowd sensing apps without any programming efforts – in addition to that, the web interface can also be used to create contents for the modules used within the apps.

It was furthermore demonstrated how crowd sensing sessions are set up in order to actually record data. In summary, it was shown that the proposed Data Collection Platform offers the means to deploy crowd sensing applications much more efficiently than other existing solutions.

How can a maximum degree of adaptability with respect to the type of deployment scenario be achieved?

With respect to the platform's adaptability, two important factors were elaborated in the scope of this chapter. Firstly, it was shown that the app's basic set of out-of-the-box application-independent features is already suitable for realizing end-user services for different types of deployment scenarios. For example, it is no problem to provide messaging services that even support location based messaging. Furthermore, web resources can be easily used in the apps – and the same is true for geographic information.

Secondly, it was shown how easy it is to extend the platform with new features. Within the Generic App Framework, adding a feature can be done by merely implementing a single interface – and in case that new feature needs to be able to update itself with new contents, by sub-classing an update channel. Additional sensors can also be easily implemented within the app framework. All that needs to be done here, is making sure that the data is being sent to the Sensor Manager's data-handling method. It was also demonstrated that new means for processing recorded data can be added by simply implementing an additional Python output worker within the Data Recording Back End.

How can the application running on the participants' smartphones be designed in a way that it provides actual value to them?

It was shown that it is possible to include modules in the crowd sensing apps that are providing actual value to end-users. As mentioned above, this can already be done with the out-of-the-box set of application-independent app modules, e.g. for providing location based services, map information (like nearby points of interest) or web contents. Furthermore, the fact that new modules can be easily added to the framework, enables deployment owners to incorporate completely new features for their users. In addition to that, the app framework's capability to update modules with new contents at any time ensures that the apps stay relevant to their users.

How can scalability of all system parts be achieved in a cost-efficient manner in order to allow for feasible truly large-scale applications?

It was shown in detail how both of the platform's back ends have been designed to be scalable from ground up. In case of the Data Processing Back End which is likely to be hosted within a self-managed server cluster for reasons of data protection, a concrete virtual machine layout was proposed that allowed for easily creating a solution for the required deployment size. Furthermore, the results of load testing benchmarks have been presented in order to provide a feeling for the hardware requirements for a given deployment scenario. Here, it was also demonstrated that even with rather limited hardware resources, the platform is able to serve a sizable user base – which is also an important fact from a financial point of view.

Likewise, with respect to costs-efficiency, it was shown that all the technologies used within the platform are freely available (in most cases, even open source). Moreover, cloud services were incorporated into the design of the Content Creation Back End in order to lower costs and still provide excellent performance at the same time.

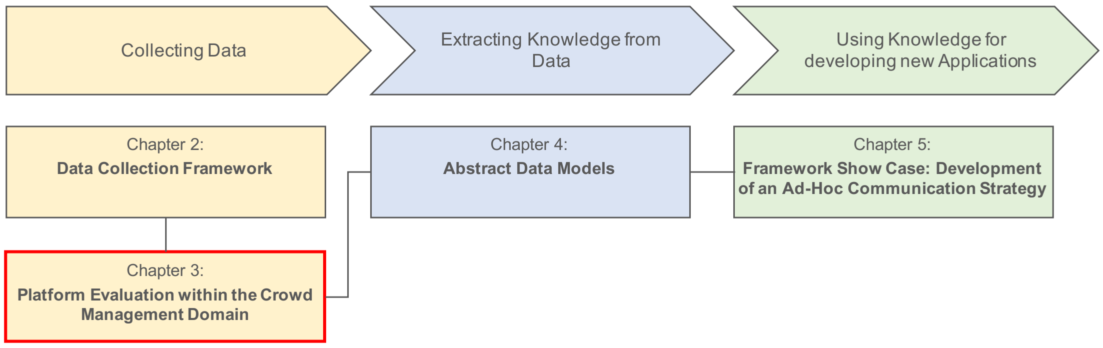
Chapter 3

Platform Evaluation within the Crowd Management Domain

"In all chaos there is a cosmos, in all disorder a secret order."

C.G. Jung

After the concept of Crowd Sensing and the proposed Data Collection Platform have been introduced in the previous chapters, this chapter's main purpose is to present an assessment of the platform under real-life, large-scale conditions. For this purpose, the crowd management domain is introduced. The results of numerous deployments within this scenario are then used as a basis for an experimental evaluation with respect to the platform's performance in the domain, the user perception and influencing soft-factors.


Collecting Data
Extracting Knowledge from Data
Using Knowledge for developing new Applications
Chapter 2: Data Collection Framework
Chapter 4: Abstract Data Models
Chapter 5: Framework Show Case: Development of an Ad-Hoc Communication Strategy
Chapter 3: Platform Evaluation within the Crowd Management Domain

3.1 Introduction

Crowd phenomena are a well-studied yet by far non-trivial example of collective human behavior [32, 33]. Understanding and controlling such phenomena has significant practical relevance in civil protection applications – managing large-scale public events and emergency evacuations are just two examples. The consequences of bad crowd management can potentially be disastrous: examples for worst case scenarios include the 1989 Hillsborough tragedy where a barrier inside a football stadium collapsed under the masses of the crowd (95 casualties, 400 injuries), the 1990 stampede in Mecca (1'426 people died inside a pedestrian tunnel) [34] or the 2010 German Love Parade disaster where 21 people were trampled to death (with more than 500 having been injured) when a mass panic broke out [35].

It therefore becomes quickly apparent that developing technologies which are able to assist with crowd management can potentially safe human lives. More specifically, such technologies should increase both situational awareness and crowd control in order to provide the means to quickly identify potentially dangerous situations and to take appropriate countermeasures.

(A) Situational Awareness

In planning and managing crowds at large scale events, situational awareness goes far beyond the mere ability to observe the area in question [36, 37]:

1. Individual observations must be combined in order to form a coherent picture that provides information about relevant global crowd parameters such as density distribution, motion directions, crowd turbulences, etc.
2. Developments and trends need to be monitored and analyzed as they evolve over time so that problems cannot only be spotted when they occur, but be foreseen in time to be prevented.
3. It must be possible to review the entire course of an event retrospectively to identify problems and irregularities and to understand their causes. While, fortunately, major accidents are rare, deviations from the plan, unexpected behavior and potentially dangerous situations are common. Understanding how and why they happened is critical for reducing the probability of major incidents in the future.

Traditionally, situational awareness during public events has been based on observations and reports from civil protection forces deployed on site. Naturally, this lead to patchy, non-real time information with limited reliability as people sporadically report via radio what they see. As a consequence, CCTV cameras are nowadays an indispensable tool, providing timely and reliable information that can be recorded and played back later on.

However, even with a large number of cameras, the available information tends to be patchy as one rarely has complete coverage of the entire operational area. Furthermore, video surveillance requires complex interpretation in terms of the overall coherent picture and global parameters – i.e. it can be very hard for an operator to quickly combine the information from potentially dozens of video feeds into an integrated situational overview. Thus, the step from parallel video streams showing isolated locations to having a reliable global picture of the overall crowd density and motion is not trivial. Deriving trends, making predictions, or finding causes for problems in such streams is even more difficult.

As a consequence, tools are required that collect, aggregate and interpret information from the entire operational area. Moreover, these tools must generate an easily understandable representation of the relevant global crowd state parameters and their evolution both in terms of time and space. One possible way of achieving this is the automatic analysis of video streams for the purpose of counting people and possibly even for tracking their movements. As an alternative – and more versatile – means for supporting crowd managers, this thesis chapter evaluates deployments of the Data Collection Platform at large scale events. During those deployments, the crowd sensed data is used to create an overview about the global crowd behavior automatically.

(B) Crowd Control

The main tool for controlling a crowd during large scale public events is the physical layout of the event-space defined and implemented in advance. It includes barriers, entrances, exits, passages etc. Ideally, such a layout is based on theoretical models and simulations [32]. In order for those models and simulations to be useful, the following three inputs are required:

1. An initial **Crowd State** – this is characterized e.g. by a crowd's size and its distribution.
2. A **Physical Space Layout** – e.g. taken directly from the event planner's blue prints
3. A certain general **Crowd Behavior** – e.g. a situation where everybody is heading towards the nearest exit or most people moving into the same direction

Using the above inputs for theoretical models, the resulting simulations can predict the evolution of global parameters such as the density distribution, evacuation time or average physical pressure within the crowd. Chapter 4 of this thesis will also also look at this modeling topic with respect to the potential impact crowd sensing applications can have on it. Provided that the input data is of a high-enough quality and a sensible set of simulations is run, this step of crowd control provides a reasonable degree of preparedness for the actual event and ensures that the most common situations have been accounted for as long as the actual crowd parameters (e.g. its size) do not differ from the assumed values.

During the event, crowd management needs to react to changes in the crowd's state and behavior in order to make sure that key parameters remain within acceptable bounds and no undesirable phenomena (e.g. panic, stampedes, blockages, jams, etc.) occur. To this end, two main types of actions can be taken: (1) dynamic changes in space configuration (e.g. closing entrances, opening new exits, setting up new barriers) and (2) issuing instructions to the crowd in order to influence its behavior. While, within certain limits given by the venue, physical space reconfiguration can be prepared beforehand (e.g. placing removable and movable barriers, or posting security personnel to redirect people) and executed effectively, issuing complex instructions to the crowd and ensuring compliance is a difficult problem. Today, it often involves "primitive" approaches such as security forces shouting through megaphones and raising improvised signposts. It goes without saying that the amount of information – and its complexity – that can be delivered this way is limited.

Going one step further, it is exceptionally difficult to deliver separate instructions to individual parts of the crowd – e.g. to direct each part to a different exit – and to supply the required background information explaining the necessity of certain measures, which is crucial to ensure compliance. However, using the Generic App Framework's location based messaging capabilities, it becomes relatively easy to deliver such targeted instructions to the crowd. Therefore, this platform evaluation chapter will also assess the feasibility of that approach.

Using the crowd management scenario introduced above as a large-scale, real-world example, this thesis chapter provides the answer to the research question *"Are deployments of the proposed architecture feasible in the real world?"* by presenting an analysis of the scenario, discussing the implementation of the deployment and showing the qualitative and quantitative results of a user study. Going more into detail, the following contributions are being made in the course of this chapter:

- A systematic, theoretical analysis regarding the feasibility of the crowd sensing approach in the crowd management domain.

- The implementation of an integrated crowd management system based on the Data Collection Platform.
- A qualitative and quantitative analysis of a user study performed in order to assess how end users and deployment owners perceive the proposed system.
- A qualitative elaboration on the lessons learned during the numerous evaluation deployments.
- A quantitative evaluation of app usage patterns leading to the conclusion that app usage can be regarded as a precursor for predicting crowd density.

In order to provide the context for this chapter, the following "Related Work" section gives an overview about existing tools for supporting crowd management at large scale events. The third section demonstrates how the Data Collection Platform is utilized for increasing the efficiency of crowd management. It furthermore elaborates on the feasibility of the proposed approach. Afterwards, the features that have been added to the platform for this specific use case are shown. Before the "Results and Lessons Learned" section provides the outcomes of the evaluation, the "Deployment History" section gives an overview about all deployments. Finally, the last part of this chapter summarizes all results and puts the chapter's contribution in context.

3.2 Related Work

Obtaining knowledge about the current size and density of a crowd is the central aspect of crowd monitoring [38]. For the last decades, automatic crowd monitoring in urban areas has mainly been performed by means of image processing [39]. One early use case for such video-based applications can be found in [40], where a CCTV camera based system is presented that automatically alerts the staff of subway stations when the waiting platform is congested. More recent research has focused on advanced computer vision based methods to analyze video feeds with the goal of detecting potentially dangerous crowd situations automatically [41–43].

However, one of the downsides of video based crowd monitoring is the fact that video cameras tend to be considered as privacy invading. Therefore, the authors of [44] differentiate between object based approaches and holistic approaches. In case of the former, actual individuals are identified and tracked in the video stream in order to calculate crowd features. Such methods are not only potentially privacy invading but also face some problems with accurately estimating the crowd density in scenarios with occlusion in high-density situations [44].

Holistic approaches on the other hand, compute more anonymous features based on the video footage – one example for this method can be found in [45] where the authors present an approach utilizing optical flow to detect potentially dangerous crowd behavior (e.g. congestions). Also, [46] presents a privacy preserving approach to video-based crowd monitoring where crowd sizes are estimated without people models or object tracking.

In [43], the authors review the state of the art of vision based systems for crowd monitoring. They conclude that the systems being currently deployed suffer from poor scalability and are unable to provide an automated situational awareness on a large scale. This point is also underlined by [47], which states that it is still difficult to fuse information gathered from multiple cameras into one coherent global picture. Finally, the obvious downside of camera-based approaches is that they are unable to capture information from outside of their fields of view [42].

A much more precise alternative to vision based approaches is the use of GPS sensors. Even in urban scenarios with high-rise buildings leading to potential problems in terms of receiving a sufficient number of satellites, GPS still manages to achieve a median location error of approximately 8 m with 95% of all location fixes being more precise than 24 m [48]. In a number of works, some people within a crowd have been equipped with dedicated GPS devices in order to analyze how the crowd as a whole behaves during an event. [49] for example used this approach to identify critical situations during the pilgrimage in Mataf (Saudi Arabia) and to re-design the infrastructure accordingly. [50] on the other hand utilizes GPS data to gain insights into the preferences of visitors at a large sports event. While handing out dedicated GPS devices to parts of a crowd has the advantage of gaining high-quality data, the obvious downside is the operational overhead for managing the devices (i.e. handing them out and collecting them afterwards). It seems much more sensible to utilize devices that are already owned by members of the crowd – the classic concept of crowd sensing.

To that end, using mobile phones as a means to gain insights into crowd behavior appears to be the ideal solution, especially given their ubiquity and the number of sensors that are built into modern phones. Here, common approaches can be generally divided into in-network-localization and on-device-localization. In case of the former, data provided by the phone network itself is being used to deduce crowd characteristics. A common feature is for example the network bandwidth usage which measures the amount of communication taking place inside a network cell. [51] and [52] follow this approach in their work, where a dataset created by approximately 40% of cell phone users in the city of Rome is used to investigate high-level crowd dynamics. The downside of this technique is that it doesn't necessarily correlate with with the actual number of people in a cell as a high bandwidth usage could also be caused by a small number of people causing a very high load on the network cell.

An alternative metric within the scope of in-network-localization is the use of call data records (CDR) which are created whenever a call is started or finished and whenever an SMS is sent or received. Amongst other things, a CDR also includes a rough location value – hence, an aggregation of all CDRs within a certain time frame can be used to deduce the crowd density distribution within that time frame. [53] and [54] use this approach to gain insights for city planning and to analyze city dynamics, respectively. The downside of this approach is that CDRs are only produced when a mobile phone is actually in use, i.e. the data points are not generated regularly. Furthermore, the precision of the location information can be as rough as 300 m [55].

The approach of creating the location information on the phones used by parts of the crowd has been followed in the scope of this thesis chapter which is based on the works previously published in [36, 37, 56–59]. Here, we propose to use smartphone apps which are deployed at large-scale events for the purpose of providing users with background information on the proceedings of the event, as data sources for a crowd sensing system. The resulting anonymized movement traces are then analyzed in real-time to compute crowd behavior metrics such as crowd density and crowd turbulences. This approach has been proven to be rather successful and led – according to the author's best knowledge – to the creation of the largest data set of its kind [58]. Details on this method shall be elaborated in the course of this chapter.

A very interesting supplementary aspect is the utilization of Bluetooth scans for gaining additional information about crowd parameters in the scope of a crowd sensing application. [60] and [61] propose an app that counts how many Bluetooth-enabled devices a user can "see" in regular intervals. The resulting information is then used to calculate the crowd density distribution. Weppner et al. [62–64] improve on that concept by not only considering the number of discoverable devices but also by computing the average observed signal strength and the variance in both the signal strength and the number of discoverable devices. In addition, the authors leverage collaboration between several close by devices. As a result of that, they achieve a 30% improvement over the crowd density classification accuracy of the approach which simply counts discoverable devices. In [64] the authors extend their methodology to also covering crowd flow in addition to crowd density.

Of course, the concept of Bluetooth scanning can also be applied in a fixed-infrastructure fashion. [65] presents a work where Bluetooth scanners were installed at strategically important locations. The recorded data was then used to calculate crowd densities and visitor flows between such locations.

3.3 Domain Requirements

Previously, the Crowd Management domain and its relevance have been introduced. In the following, it is shown how the Data Collection Platform is utilized within such scenarios in order to help organizers of large scale events with crowd management. On a high level, the platform's components are used for the following tasks:

1. The **Generic App Platform** is used to create event-specific apps which are downloaded by event visitors for the purpose of getting access to background information on the event. Hence, the apps are containing information about event proceedings, geographic layout of the venue, etc. – they also contain social media contents and possibly gamification elements. This feature set provides a substantial value proposition for event visitors and therefore makes sure that the required number of users is actually achieved. An overview about domain-specific modules that have been added to the Generic App Platform is given in 3.3.4.

   From a domain point of view, the event apps are turning visitors' smartphones into a distributed sensor network recording data about the crowd's current state with respect to e.g. its density distribution. Hence, they act as data sources for the crowd sensing system. Furthermore, the Generic App Framework's capability to receive messages in a context specific manner allows event organizers to provide visitors with relevant information and advice during potentially dangerous situations.

2. The **Content Creation Back End** enables event organizers to design and administrate their apps without any actual programming efforts. It also provides the tools for creating app contents (like points of interest or event schedules), for planning crowd sensing sessions and for sending messages.

3. The **Data Processing Back End** is the component that receives and processes the data contributed by the event visitors. Its output is the real-time visualization of the currently prevailing crowd conditions. Consequently, it provides event organizers with situational awareness.

As it was already mentioned, the two central aspects of efficient crowd management are *situational awareness* and *crowd control*. The following two sub-sections will present how the Data Collection Platform assists event organizers with each of those aspects. Based on this, section 3.3.3 then demonstrates the feasibility of the proposed approach.

3.3.1 Situational Awareness by Means of Crowd Sensing

The concept of providing situational awareness to security forces and organizers of large events by means of crowd sensing was researched in collaboration with Wirz et al. [36, 37, 56–59]. Specifically, my contribution was the development and evaluation of the Data Collection Platform [56, 57] while the team around Martin Wirz from ETH Zurich was responsible for creating the visualization of the recorded data in a way that would provide relevant insights for crowd managers [36, 37, 59]. As this thesis chapter's focus is the evaluation of the Data Collection Platform during real-life, large-scale scenarios, I will only summarize the data visualization in order to provide the reader with a general understanding of the approach. This summary is based on [59]. A very thorough elaboration from a more theoretical data science point of view on this topic can be found in [36].

The core principle of the proposed approach is to use the crowd sensing data to create a global overview about the current crowd behavior on a map thereby providing an easy-to-interpret visualization which allows for quickly identifying potentially dangerous situations. As a first step towards this goal it is obviously important to define a set of relevant crowd metrics that enable the assessment of the criticality of the current situation. The most apparent measure seems to be the **crowd density** as it has been shown that most stampedes occur in high-density crowds [32]. Furthermore, [66] and [67] have identified **crowd velocity** or crowd flow as an important indicator of critical situations. In fact, Johansson et al. have identified a relation between crowd flow and crowd density [66] – it was shown that a critical density has been reached as soon as the flow of a crowd breaks down. The same work also showed **crowd turbulences** to be of extreme relevance. This term describes irregular flows characterized by random displacement into all possible directions – in these critical conditions, members of the crowd are unable to control their motion and are pushed forwards and backwards by others. [68] and [66] furthermore suggest to quantify the criticality of a crowd situation by a metric they refer to as **crowd pressure** which is defined as the local pedestrian density multiplied by the variance of the local velocity of the crowd.

After having identified the relevant metrics, it is time to look at how these measures can be intuitively visualized in a way that does not require any special training and can be understood at a glance. It was decided to use heat maps to visualize the crowd behavior as this representation is very versatile and intuitive. In general terms, a heat map is a graphical representation of spatial data where regions are colored according to measurement values found at the specific location.

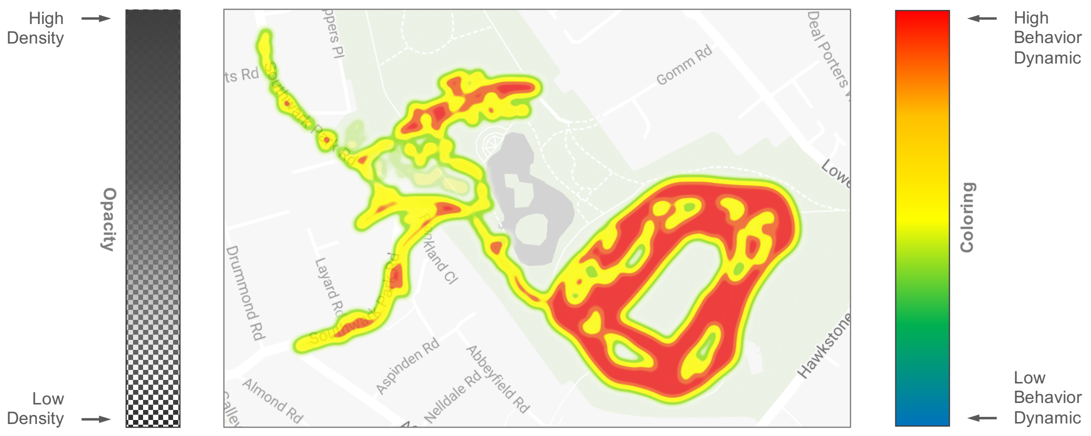


FIGURE 3.1: Heat map visualization principle

The proposed heat map visualization is explained in Fig. 3.1. It can be seen that the heat map is mainly controlled by two variables: opacity and coloring. The crowd density is expressed as the heat map's opacity, meaning that areas with a very low density are almost transparent while areas with a very high crowd density are opaque. The coloring of an area depends on the value of the specific behavior dynamic with respect to the metric that is currently visualized (e.g. crowd pressure, crowd turbulences, etc.). Here, a very warm color signals areas of high behavior dynamic, whereas areas colored in colder shades express low behavior dynamic. By making the crowd density part of the visualization of other crowd metrics, it is ensured that areas where only a small number of data points contribute to the calculation of the metric are visualized in a less prominent fashion than areas where a larger number of data points was available. If users of the system decide to only visualize crowd density, both the opacity and the color values are representing crowd density meaning that areas of lower density are displayed in a colder shade and less opaque than areas with higher density.

As already mentioned, the input data for creating the heat map visualization is recorded with the help of event specific apps created with the Generic App Framework. More specifically, within this specific use case, the input data only consists of location data. App users are asked if they would like to contribute their anonymized location data upon entering the premises of the event – if they agree, their smartphones send out the data for the entire duration of the event or until they leave the event area again. This data recording and processing process has been outlined in the previous chapter.

Given that within this real-time scenario, the goal is to determine crowd conditions at a time t, all location updates within $[t - \delta, t]$ are considered. Given that the crowd sensing app's upload interval was set to 60 seconds within the scope of the evaluated deployments, δ was also set to 60 seconds. In case one user contributed multiple location updates within this time frame, only the latest location is being used. In order to calculate a user's walking speed, the distance travelled within $[t - \delta, t]$ is determined and divided by δ. For calculating the distance between two GPS points, the Haversine function is being used. Finally, a user's heading direction θ is determined by examining the angle between the user's most recent location updates. In summary, we therefore have a set U of active users u_1, ..., u_N for each time t. Each user u_i then has an associated location $X_{i,t}$, velocity $v_{i,t}$ and heading direction $\theta_{i,t}$. These basic measures serve as input for calculating the metrics of crowd behavior introduced above. Details about the way these metrics are calculated are given in the following.

Crowd Density

The crowd density estimation is calculated by applying a Kernel Density Estimation (KDE) according to [69] from the user's location information at any given time t. KDE is a non-parametric way of creating a smooth map of density values in which the density at each location reflects the concentration of sample points. Hereby, each sample point contributes to the density estimation based on the distance from it. By using a Gaussian kernel K, the density estimation $\hat{d}$ at each location X is given by:

$$\hat{d}(X,t) = \frac{1}{N \cdot h} \sum_{i=1}^{N} K\left(\frac{X - X_{i,t}}{h}\right) \tag{3.1}$$

Here, h refers to the bandwidth which is an application dependent smoothing parameter. Furthermore, the Gaussian kernel function K is given by:

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}u^2\right) \tag{3.2}$$

The crowd density heat map is now created by calculating the density values $\hat{d}(X,t)$ for each location. The results are then mapped either only to an opacity value or also to a color value – depending on whether or not the heat map is supposed to display another behavior metric in addition to the crowd density.

Crowd Velocity

The local crowd velocity can be considered as the weighted average velocity of each user near the location. The individual speed values are weighted with a Gaussian weighting scheme based on their distances to that location. Taking this as a basis, the formula for calculating the crowd velocity estimation $\hat{v}(X,t)$ at location X and time t is given by:

$$\hat{v}(X,t) = \frac{\sum_{i=1}^{N} v_{i,t} K\left(\frac{X - X_{i,t}}{h}\right)}{\sum_{i=1}^{N} K\left(\frac{X - X_{i,t}}{h}\right)} \tag{3.3}$$

Again, $K(u)$ refers to the Gaussian kernel specified in equation 3.2 that is applied to weigh each of the users' speed values $v_{i,t}$ at time t. The resulting crowd velocity value is then mapped to a color value for the heat map visualization.

Crowd Turbulence

The crowd turbulence $\hat{c}(X,t)$ during time t is calculated based on the variance of the users' heading directions at location X. More specifically, the circular variance $z_{i,t}$ [70] is weighted again using the Gaussian kernel K specified in equation 3.2. This results in the following formula for calculating the crowd turbulence value:

$$\hat{c}(X,t) = 1 - \left| \frac{\sum_{i=1}^{N} z_{i,t} K\left(\frac{X - X_{i,t}}{h}\right)}{\sum_{i=1}^{N} K\left(\frac{X - X_{i,t}}{h}\right)} \right| \tag{3.4}$$

Here, $z_{i,t} = e^{i\theta_{i,t}}$ and $\theta_{i,t}$ describes the heading angle of user i at location X_i during time t. Again, the resulting crowd turbulence value is mapped to a color value for the heat map visualization.

Crowd Pressure

Helbing et al. define crowd pressure in [68] according to the following formula:

$$P(X) = \rho(X) \cdot \mathrm{Var}_X(\vec{v}) \tag{3.5}$$

Here, ρ is the local pedestrian density and $\mathrm{Var}_X(\vec{v})$ the local velocity variance. While the former has already been defined in equation 3.1, the latter is given by:

$$\hat{\mathrm{Var}}_{X,t}(\vec{v}) = \frac{\sum_{i=1}^{N} \left|\vec{v}_{X_{i,t}} - \langle\vec{v}\rangle_{X,t}\right| \cdot K\left(\frac{X - X_{i,t}}{h}\right)}{\sum_{i=1}^{N} K\left(\frac{X - X_{i,t}}{h}\right)} \tag{3.6}$$

with $\langle\vec{v}\rangle_{X,t}$ simply being the crowd velocity $\hat{v}(X,t)$. Again, K is the Gaussian kernel used to weigh the individual values. Using equations 3.5 and 3.6 as a basis, the crowd pressure estimation $\hat{P}(X,t)$ at the location X during time t is therefore defined as:

$$\hat{P}(X,t) = \hat{d}(X,t) \cdot \hat{\mathrm{Var}}_{X,t}(\vec{v}) \tag{3.7}$$

Corresponding to the other metrics above, the resulting crowd pressure values are mapped to color values which are then used to render the heat map.

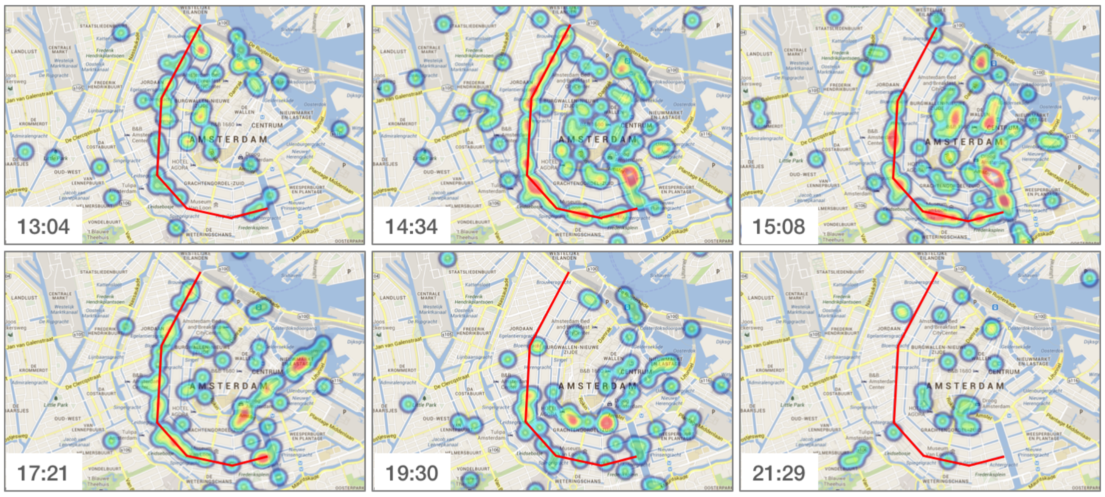


Figure 3.2: Sample heat maps visualizing crowd density during a large event in Amsterdam at several stages throughout the day

The above Fig. 3.2 gives an example of how the proposed heat map visualization can help with providing situational awareness throughout an entire event. In this instance, the Data Collection Platform was deployed during a large city-wide festival in Amsterdam. Around 14:00 the highlight of the event took place: a parade through one of the city's many channels (marked with a pink line on the maps). At 15:00 a series of concerts took place at several locations throughout the city. Finally, at 19:00 the last of the concerts was over and people started to flow into the surrounding bars. Even without any domain knowledge it becomes apparent that the heat map visualization helps enormously with quickly identifying the most crowded areas at each event phase. The responsible safety and security staff can therefore quickly react on changing crowd conditions – e.g. by consulting CCTV camera footage of the affected area or by dispatching members of staff.

3.3.2 Crowd Control by Means of Context Specific Messaging

As it was already mentioned in the introductory section, situational awareness is only one part of the crowd management domain. In fact, without being able to act upon the information made available by a crowd sensing system, its overall value proposition is greatly reduced. Therefore, the event-specific apps deployed in the crowd management scenario all have the messaging module enabled which allows deployment owners to get in touch directly with users of their apps.

In this context, the concept of location based messages is especially important. These messages can be targeted at a geographic area of arbitrary shape and size. Only people who are located within that area are receiving the message – this characteristic makes them ideal for "steering" the crowd to avoid potentially dangerous situations.

In combination with the information gathered via the crowd sensed data, location based messages realize a form of closed feedback loop. Crowd managers can identify potentially dangerous regions and immediately send instructions to people affected by the situation. This can happen in a context specific fashion: in case of a congestion for example, people approaching the area from the north can be sent different re-routing advice than people who are approaching the area from the south.

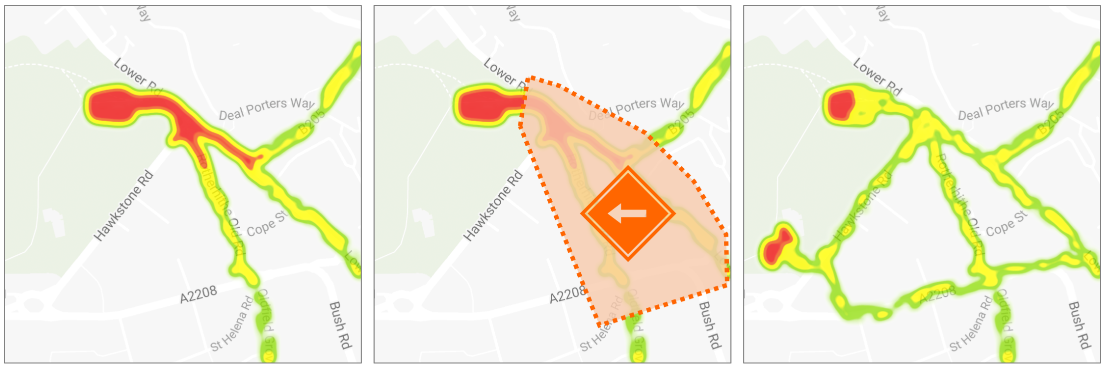


FIGURE 3.3: Location based messages as a means for crowd control
(Conceptual Drawing)

Figure 3.3 underlines this principle again. In the left part, it can be seen how a large congestion is blocking most of the north east corner of an event venue. This congestion is clearly caused by people trying to enter the venue. In the middle part of the figure, a location based message is sent to the major ingress routes advising people to also consider using the southern entry. In the right part of the figure, the large congestion has been cleared as people are now also moving towards the alternative entry which was less obvious to them due to their arrival route. It can also be seen that the areas expressing high crowd density are now a lot smaller compared to the initial situation.

3.3.3 Feasibility of the Approach

After the previous two sections have shown how the Data Collection Platform can be used to assist with crowd management, the following section focuses on demonstrating the feasibility of this approach. In the first part, it is shown that having a fraction of the event visitors voluntarily contribute crowd sensed data is enough to provide meaningful insights into the general crowd behavior. Furthermore, the impact of the percentage of participating visitors on the system's overall "resolution" is discussed. The second part of this section deals with the topic of communication load. Here, it is shown how much traffic the Data Collection Platform is causing per user in order to provide a feeling for its impact on the users' data plans.

Part 1: Required Number of Users

In simple terms, density estimation means that, for any given area within the observed space, a system can provide the percentage of visitors who are located in that area. The proposed crowd sensing approach determines this percentage based on the sample of visitors who are running our app: if x percent of them are within a given area, the same is assumed to be true for all users. Essentially, this process is identical to performing a statistical sampling which is a common technique e.g. in opinion surveys. Here, the relationship between sample size – which in the crowd management case equals the number of visitors using the crowd sensing app – and the accuracy of the estimate is well understood.

For a sufficiently large sample size n, the distribution of a population proportion will be closely approximated by a normal distribution and therefore the margin of error for the estimation of a population proportion ME_p can be calculated with the Wald method using the formula

$$ME_p = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \cdot z_{\alpha/2} \tag{3.8}$$

where $\hat{p}$ is the sample proportion, n is the sample size and $z_{\alpha/2}$ is the $100(1-\alpha/2)$th percentile of the standard normal distribution (i.e. the confidence level). However, the downside of equation 3.8 is that it doesn't take the ratio between the sample size and the size of the overall population into account – a fact which can be neglected when investigating "sufficiently large sample sizes" in scenarios with large populations. In case of large scale events however, the ratio between the number of visitors participating in crowd sensing and the total number of visitors does play a role since the total population may only be as small as 100'000 (or less). Therefore, a slight adaptation of the Wald method is used for determining the margin of error of a population proportion:

$$ME_p = \sqrt{\frac{\hat{p}(1-\hat{p})}{n} \cdot \varepsilon} \cdot z_{\alpha/2} \qquad \text{with } \varepsilon = \begin{cases} 1 & \text{, if } pop \text{ is unknown} \\ \frac{pop-n}{pop-1} & \text{, otherwise} \end{cases} \tag{3.9}$$

Here, the factor ε has been introduced. It is set to 1 in case the size of the population *pop* is unknown or not of relevance. In all other cases, ε weighs the term below the square root according to the ratio between the sample size and the population size. Assuming a fixed value of n, this factor ensures that the margin of error is larger in case of greater population sizes than it is in case of smaller population sizes as the available samples become less representative of the whole. In the extreme case where $n = pop$ (i.e. all event visitors are using the crowd sensing app), the margin of error is 0 – in other words, the measurement is perfectly representative.

In the scope of this evaluation, the target is a confidence level of 95 % (i.e. $z_{\alpha/2} = 1.96$). Since we don't know the actual sample proportion, we must assume the worst case percentage of 50 % (i.e. $\hat{p} = 0.5$). Thus, if equation 3.9 results in $ME_p = 0.01$, it means that with a 95 % probability, the true number of people within a given area will be within ± 1 % of the percentage estimated from the data provided by the app users. This leads directly to the observation that in a scenario with a total population of 100'000 people and a margin of error of 1 %, the estimation will deviate from the reality by ± 1'000 people.

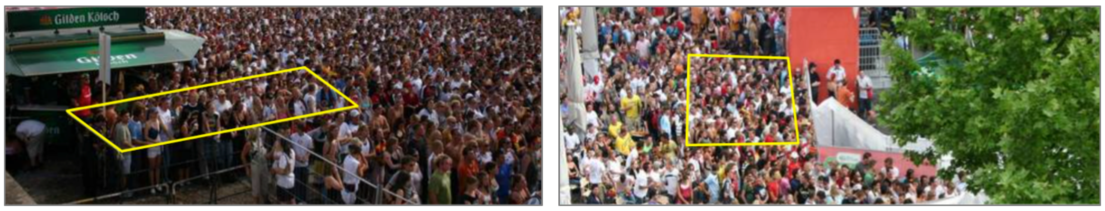


Figure 3.4: Examples of different crowd densities during public viewing events. Left: 3.8 persons per m^2, right: 5.0 persons per m^2. (Source: [71])

Besides the margin of error, the second critical aspect for establishing the feasibility of the proposed crowd sensing approach are the types of density effects that are relevant for event planning and crowd management. According to [71], large scale events are planning with an average crowd density of 2 persons per square meter. In some countries (e.g. Germany) this value is even part of the law. While 4 persons per square meter are considered as very crowded, [71] furthermore states that in reality the crowd density hardly ever surpasses 6 persons per square meter (which is considered as critically crowded). In order to provide a feeling for those density values, Figure 3.4 shows two examples of crowd densities recorded during public viewing events.

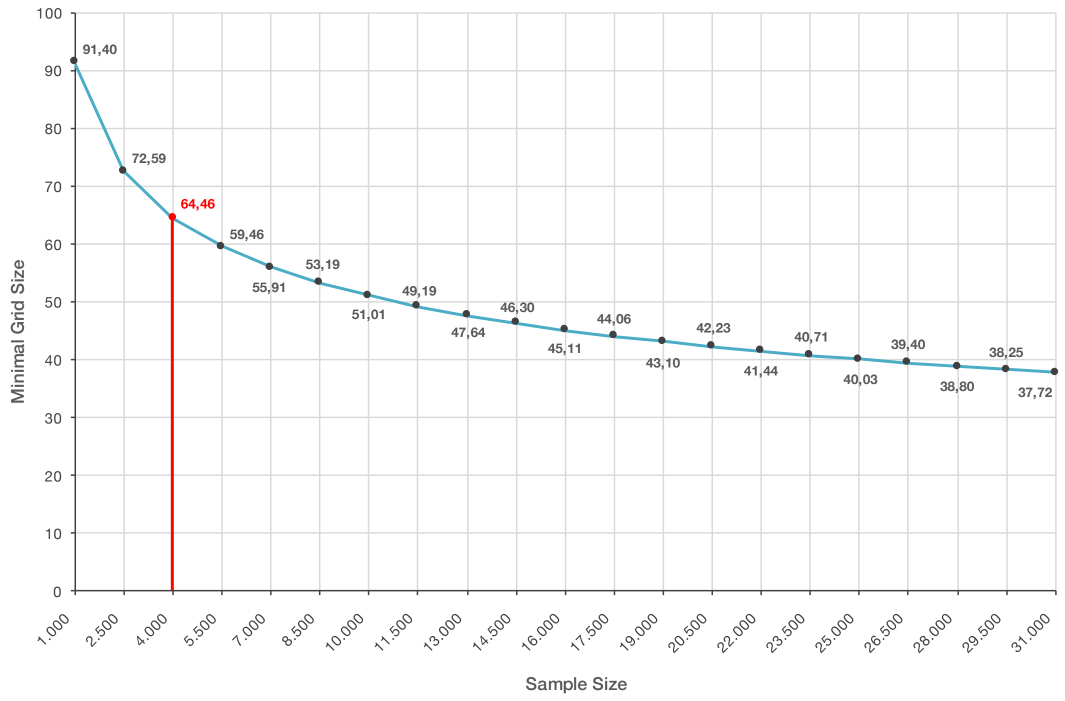
Consequently, within the crowd management domain, the Data Collection Platform needs to ensure that crowd managers are able to reliably distinguish between a number of different density states: empty, 2 people/m^2, 4 people/m^2 and 6 people/m^2. Now, the connection between these considerations and the margin of error described above is to be found in spatial resolution. The margin of error can be given as absolute number of people – integrating the density over an area also results in an absolute number of people. The larger that area, the larger not only the absolute number of people but also the absolute number of people by which the relevant density states differ.

In order to provide a deeper understanding for this connection, a concrete deployment of the Data Collection Platform is taken as an example. During that deployment, a total of 28'000 event visitors were using the crowd sensing app, with about 4'000 simultaneous uploaders at peak times. During those peak times, the total population of the event venue consisted of roughly 300'000 visitors. Taking equation 3.9 and setting $n = 4'000$ and $pop = 300'000$ leads to a margin of error of 1.54 % or 4'620 people ($0.0154 \times 300'000$).

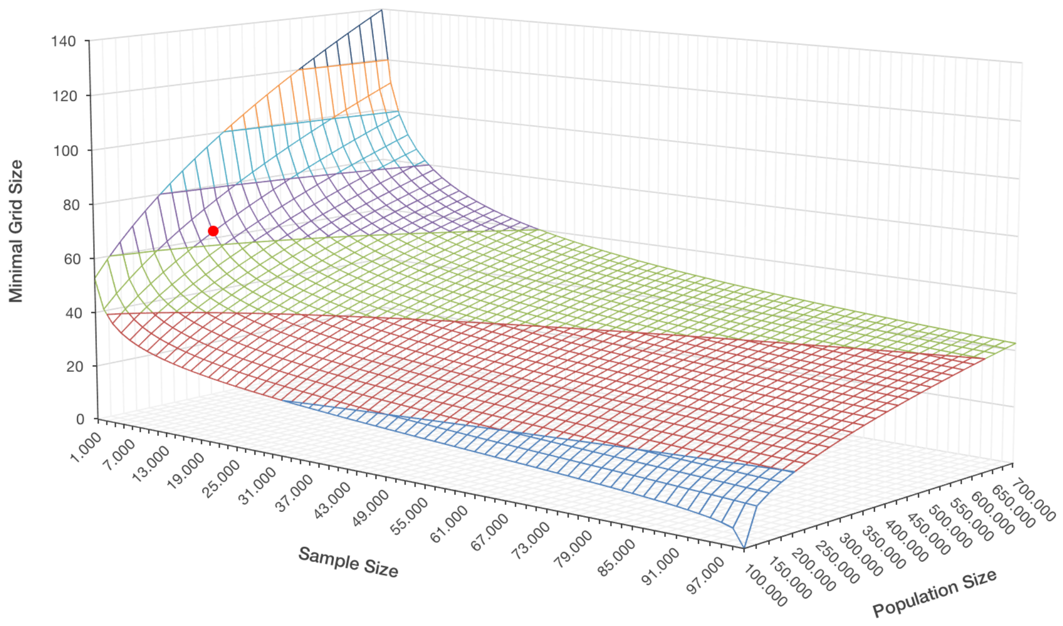
If one now considers a grid with rectangular cells of size 75 m by 75 m laid on top of the event venue, it means that a cell expressing a crowd density of 2 people/m^2 would contain 11'250 people, a cell with a crowd density of 4 people/m^2 would contain 22'500 people and a cell with a (critical) crowd density of 6 people/m^2 would contain 33'750 people. Based on this 75 m grid and the previously calculated margin of error, it can be concluded that within this specific deployment, the system allowed for distinguishing the relevant states as the distance in terms of number of people per cell between each state (11'250) is greater than double the margin of error ($2 \times 4'620 = 9'240$). Consequently, the crowd sensing application could provide crowd managers with reliable information about crowd behavior with a precision of at least 75 m.

When looking at the numbers, it is clear that the quoted spatial resolution is lower than the ideal value that could be achieved in that deployment. In theory, the size of the grid cells could be reduced to a level where each density state is separated by exactly twice the margin of error. In practice however, it is advisable to build in some buffer – within the scope of this work, a buffer of 10 % was chosen. To provide a better feeling about the impact the parameters n and pop have on the spatial resolution, Fig. 3.5 gives an overview about their interdependence. In the figure, the scenario quoted above is visualized in the upper half while the lower half visualizes the spatial resolution for various population sizes. In both cases, the situation that was present during the exemplified deployment, is highlighted with a red marker – in that instance, the maximal spatial resolution that could be achieved by the system was 64.46 m. A further very relevant insight of Fig. 3.5

is also, that after a certain point, an increase of the sample size n does not lead to a meaningful gain in spatial resolution: e.g. in the sample deployment, a 300 % increase from n = 10'000 to n = 30'000 would only lead to an increase in spatial resolution of roughly 25 %.



(A) Fixed population size (*pop* = 300'000)



(B) variable population size

FIGURE 3.5: Maximal achievable spatial resolution with respect to crowd density estimations as a function of the sample size at fixed and variable population sizes.

In conclusion, it can be said that even with a comparatively small number of participants, the proposed crowd sensing approach is in fact able to provide representative estimates of the crowd density distribution at an event venue with a sensible spatial resolution. This leads directly to the indication that the other metrics of crowd behavior (e.g. crowd pressure, etc.) calculated by the system are also representative because they are strongly connected to the crowd density. In fact, it can even be said that the higher the crowd density, the more representative the other metrics will be. This is underlined by the work of Moussaid et al. who showed that pedestrians in crowds are very likely to mimic the behavior of their neighbors, e.g. by adjusting their walking speed and direction [72].

Part 2: Communication Load

With respect to communication load, it needs to be ensured that the crowd sensing app doesn't cause too much data traffic. If visitors have the feeling that their data plans are being consumed disproportionately, they will stop using the app – this in turn would mean the end of the crowd sensing system. As it was shown previously, there are two types of data streams which need to be considered:

1. Sensor data being uploaded to the Data Processing Back End – in case of the crowd management scenario, this consists only of location data.
2. Content update calls being performed in regular intervals – this also includes the download of contents.

With respect to the the content update calls, it is very hard to quantify the actual data traffic as it depends completely on the specific deployment. Some deployment owners may never change the layout, contents and settings of their apps after they have been created – which would result in zero traffic as all the initial contents are downloaded along with the app from the app stores – others may update their apps' contents frequently with high resolution images thereby causing more traffic. The same argument can be made for the traffic caused by messages: some deployment owners may decide to heavily rely on multimedia messaging for improving the visitors' overall experience while others may only send a few text messages for crowd control. In summary, the bandwidth of possible scenarios is too wide to provide a precise number with respect to content updates.

What can be measured in terms of update calls however, is the base traffic caused by the regular update polls. While this traffic does of course also depend on the configuration of an app with respect to the number of modules requiring updates and their respective update intervals, an analysis of the app configurations during all evaluated deployments (see section 3.4 for details) led to an average traffic of 1.1 KB per update call – this number includes requests of all update channels and the corresponding responses containing the HTTP status 304.

With respect to the uploads of sensor data to the Data Processing Back End, the situation is a lot clearer: each minute, two GPS location fixes are being sent out and a confirmation from the back end is received. This results in an average traffic of 1.0 KB per upload call.

	Sensor Data Uploads	**Content Update Calls**
Traffic per Call	1.0 KB	1.1 KB
Traffic per Hour	60.0 KB	66.0 KB
Traffic per Day	1.4 MB	1.6 MB

Table 3.1: Communication load caused by the Data Collection Platform

Table 3.1 summarizes the above findings in a more compact form. It can be seen that even in case of 24 hour deployments, the base traffic caused by the Data Collection Platform is very low – especially when comparing it to the amount of data consumed by popular social media apps. As mentioned previously though, this table disregards the traffic caused by contents sent out by deployment owners as it is unpredictable and depends heavily on the nature of the deployment. However, since this sort of additional traffic can be considered as adding value to the visitor's experience during an event, it seems plausible that users will not perceive it as a burden in the same way that they would consider a large amount of base traffic – which adds no immediate value from a user's point of view – as an argument against using a crowd sensing app.

3.3.4 Domain-Specific Framework Features

Chapter 2.4 already presented the Generic App Framework's scenario-independent features. It was shown that using those app modules, it is possible to create crowd sensing apps that provide their users with an actual benefit. For example, context specific messaging services or map-based information services can be realized out of the box.

However, in order to evaluate the Data Collection Platform's performance within the crowd management domain under ideal circumstances, it was decided to implement a number of app modules specifically for this evaluation scenario. Table 3.2 provides an overview about these domain-specific features.

Module Name	**Module Description**
Event Calendar	The event calendar module is intended for events spanning a duration of several days or for city administrations managing all their events with a single app. In a calendar view, the users get to pick a day and are then presented with details about all relevant events during that day.
One-Day Event Overview	The one-day event module is intended for short events. Users get an overview about the event proceedings on one screen.
Checklist	The checklist module allows deployment owners to publish information about items that visitors should take along while being at the event. It is especially useful for managing festivals where people might spend several days in tents.
RSS Reader	The RSS module makes RSS news feeds available to users of the crowd sensing apps.
Twitter	As many events communicate via Twitter, the respective module allows for making selected feeds available via the crowd sensing app. This module also acts as an alternative information channel.
Mobile Heat Map	The mobile heat map module makes a live crowd density heat map available to app users in order to provide them with a situational overview.
Badge Collector	The badge collector module is a gamification feature. It allows deployment owners to define virtual badges – e.g. for visiting each concert stage. Whenever users fulfill a badge's requirements, it is added to their virtual badge collection. Badges can also be shared via Facebook.
Friend Finder	The friend finder module allows users to share their location with friends for a configurable amount of time. While the friendship connections between app users are made using the Facebook API, no location data whatsoever is being sent to Facebook.

Table 3.2: Overview about crowd management app framework features.

This set of domain-specific features was defined in cooperation with the organizers of the events used for the platform's evaluation (see the next section for details). It therefore covers all the needs that those event organizers saw as most pressing. Section 3.5.4 will investigate how those features were perceived by the actual users of the apps.

In general, the majority of the domain-specific app modules have the goal of making visitors aware of an event's proceedings and its geographic layout and to keep them informed about current news. Consequently, the main motivation for people to download the app, is to get access to this information in order to be prepared and up to date. The Badge Collector and Friend Finder modules on the other hand are geared towards increasing app uptake by means of gamification.

With respect to the Mobile Heat Map module, it must be said that it was by far the most divisive feature amongst deployment owners. While some security forces categorically ruled out the option to deploy this module due to fear of terror threats, others wholeheartedly embraced it. In deployments where the module was activated, it turned out to be one of the most popular app features (see section 3.5.4 for details) amongst visitors. This serves as an example that even the direct output of a crowd sensing application can already be of interest to the general public.

3.4 Deployment History

The very first deployment of the Data Collection platform took part during the 2011 Notte Bianca – an annual arts and music event attracting approximately 100'000 visitors – in the city of Valletta, the capital of Malta. The original purpose of this deployment was merely to check the basic functionality of the platform and see if its use would be feasible at all within the crowd management domain. With not even 300 visitors using the event app, the user base was very small. Nevertheless, a number of GoPro cameras was installed at strategically important locations to record ground truth data which could later on be used to check if the recorded data showed any correlation to the crowd behavior at all. Furthermore, a survey amongst the members of the committee organizing the event was performed in order to gather first-hand feedback.

The results of this deployment were surprisingly positive: despite the low app penetration of much less than 1 %, the estimated crowd density information was quite representative of the actual crowd behavior – at least on a qualitative level. This was also underlined by the surveys which showed that the approach was well received by the civil protection forced (the Civil Protection Department of Malta and the Ministry of Home Affairs). The survey participants especially emphasized the importance of real-time crowd density information for identifying potentially problematic areas.

Based on those positive results, it was decided to really use the crowd management domain as evaluation scenario and to develop specific app modules for it. Subsequently, a presentation about the platform was given to a panel of British policy makers in the field of public safety consisting of the City of London Police, the British Transport Police, the London Ambulance Services, Wembley Stadium and Transport for London. The goal of this presentation was to find further candidates for deployments which would be willing to support the research team with feedback and which would have the ability to deploy and evaluate the application at large scale events. As a result of this, the Lord Mayor's Show – a yearly procession through the City of London with more than 800 years of tradition and around 500'000 visitors – emerged as a prime candidate.

The first deployment at the Lord Mayor's Show in November 2011 was the first occasion during which the Data Collection Platform was deployed with its complete feature set. The data contributed by more than 800 visitors built the basis for the first series of publications on this topic [36, 37, 59]. Again, the feedback was so positive that the event has been using the Data Collection Platform ever since.

In the wake of the feedback sessions and evaluation panels of the Lord Mayor's Show, a number of new deployment opportunities came into existence with the 2012 West End Live Festival in London and the 2012 New Year's Eve celebrations in Zurich being relatively low-key examples. On the other hand, the deployment during the 2012 Olympics in London was a major opportunity from an image point-of-view. However, since the crowd sensing app was not the official Olympics app but only a smaller offering from the Westminster City Council with modest PR efforts, the resulting user count was far less than one would have hoped for (see table 3.3 for details).

The first major breakthrough with respect to user uptake was the deployment during the coronation of the Dutch King in the spring of 2013. During this setup which was organized and backed by the Dutch National Police, more than 70'000 people (more than 10 % of all visitors) downloaded the event-specific app. This achievement was mainly accomplished by a large and well-thought-out PR campaign planned by the Dutch Police. However, the actual recording of data was prevented by the intervention of a major Dutch telecommunications provider, who claimed fears of network congestion on the highest political level. While those fears were cleared eventually, it was too late to actually perform the recording.

Therefore, the real breakthrough in terms of recorded data volume was the installation at the 2013 Zurich Festival [58]. Here, of the roughly 2 million visitors, more than 56'000 downloaded the event app and more than 28'000 of them contributed data. To the best of my knowledge, this led to the creation to the largest crowd sensing data set of its kind which is also relied on heavily in the course of this thesis.

The Zurich Festival deployment was followed up by a number of Dutch event deployments which included the 2013 Amsterdam Gay Pride and the 2014 King's Day celebrations. An overview about all evaluation setups is given in Table 3.3. Here, it can be seen that in total more than 50'000 people contributed data during the course of all evaluation events with more than 100'000 people having used event apps based on the Generic App Framework. Please note that a sizable part of the deployments are labelled as "minor deployments" where the research team had no control about app distribution and merely provided the technology for interested partners. Consequently, no reliable numbers with respect to total visitor count and number of app downloads were available since this information was not disclosed. Only the "major deployments" were under full control of the researchers. Also, the number of visitors for each event (where available) are estimates as all of them were non-ticketed events without access control – hence, no overall ground truth data was available.

Event Name	**# Visitors**	**# App Downloads**	**# Data Contributors**
		Major Deployments	
Notte Bianca, 2011	≈ 100′000	≈ 1′000	≈ 340
Lord Mayor's Show, 2011	≈ 500′000	≈ 3′000	≈ 830
Lord Mayor's Show, 2012	≈ 500′000	≈ 1′000	≈ 920
Zurich New Year's Eve, 2012	n.a.	≈ 5′000	≈ 3′000
Dutch Coronation, 2013	≈ 750′000	≈ 70′000	n.a.
Zurich Festival, 2013	≈ 2′000′000	≈ 56′000	≈ 28′000
		Minor Deployments	
West End Live Festival, 2012	n.a.	n.a.	≈ 950
Westminster Olympics App, 2012	n.a.	n.a.	≈ 2′300
Vier Daagse Feesten, 2013	n.a.	n.a.	≈ 2′100
Amsterdam Gay Pride, 2013	n.a.	n.a.	≈ 1′800
Tilburgse Kermis, 2013	n.a.	n.a.	≈ 1′500
Koningsdag, 2014	n.a.	n.a.	≈ 4′600
Vier Daagse Feesten, 2014	n.a.	n.a.	≈ 2′200
Leids Ontzet, 2014	n.a.	n.a.	≈ 1′700

Table 3.3: Overview of the Data Collection Platform's evaluation deployments between 2011 and 2014.

3.5 Results and Lessons Learned

After the evaluation scenario has been described in detail, the following sub-sections present the results of the evaluation from an application point of view (for technical benchmarks, see sections 2.5.2 and 2.5.3, for theoretical elaborations on the feasibility of the approach, see section 3.3.3). First, it is shown how the Data Collection Platform has managed to provide insights into crowd behavior from a practical perspective. Afterwards, the outcomes of a user study are presented by looking at app users and deployment owners separately. Subsequently, an experience report about external factors that influenced the deployments is given in order to provide an understanding about non-technical facts that may impact a crowd sensing deployment and therefore have to be accounted for. The final sub-section then analyses how event visitors actually interacted with the crowd sensing apps in order to assess if the feature set has been chosen sensibly.

3.5.1 Insights into Crowd Behavior

Section 3.3.3 already showed that the proposed approach to estimate crowd behavior by means of crowd sensing is feasible from a theoretical point of view. Furthermore, an evaluation of the approach from a data science perspective can be found in [36]. In the following, the practical side of this topic is elaborated on a qualitative level by presenting actual findings from the evaluation deployments.

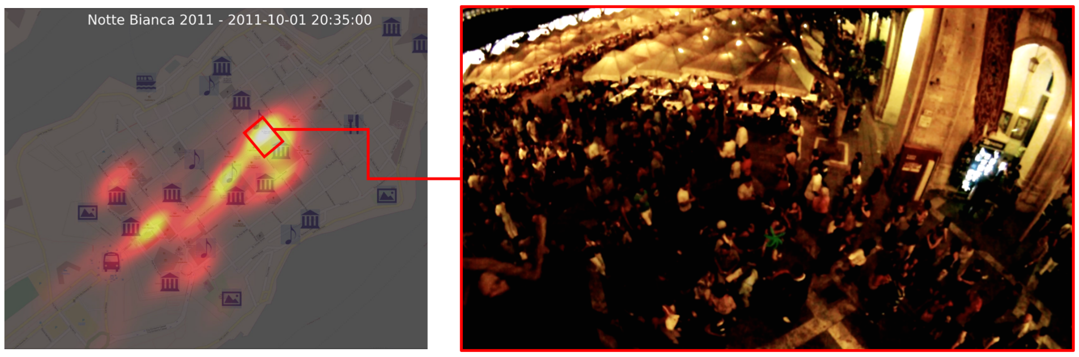


FIGURE 3.6: Crowd density estimation compared with ground truth video footage recorded during the 2011 Notte Bianca festival in Valletta, Malta

The general consensus of all post-event evaluation discussions with the involved stakeholders was that the crowd sensing approach provided a good representation of the situation on the ground. Furthermore, the visualization gave valuable insights into the temporal and spatial evolution of the crowd, thereby facilitating situational awareness, prediction and post-event analysis. Details on this user feedback can be found in section 3.5.2. The most surprising insight was that even in cases of extremely low app penetration, the resulting visualizations were considered as useful by the involved safety and security experts. Figure 3.6 shows an example of such a deployment: while only about 0.3 % of the event visitors contributed data to the crowd sensing system, the resulting insights were generally representative of the overall situation.

Of course, deployments with a larger app penetration provided a lot more meaningful insights. The best example for this was the aforementioned deployment at the Zurich Festival in 2013 where a total of 28'000 people contributed data (≈ 1.4 % of the overall visitors). Here, it was possible to analyze crowd behavior with a much larger spatial resolution. Figure 3.7 shows the development of the estimated crowd size (indicated by the number of app users) in the micro-scale within one particular area over the course of 15 hours. It can be clearly seen that the estimated crowd size correlates with real-life events taking place in that area. The peak is reached just before the fireworks display (the highlight of the festival) starts at 22:30.

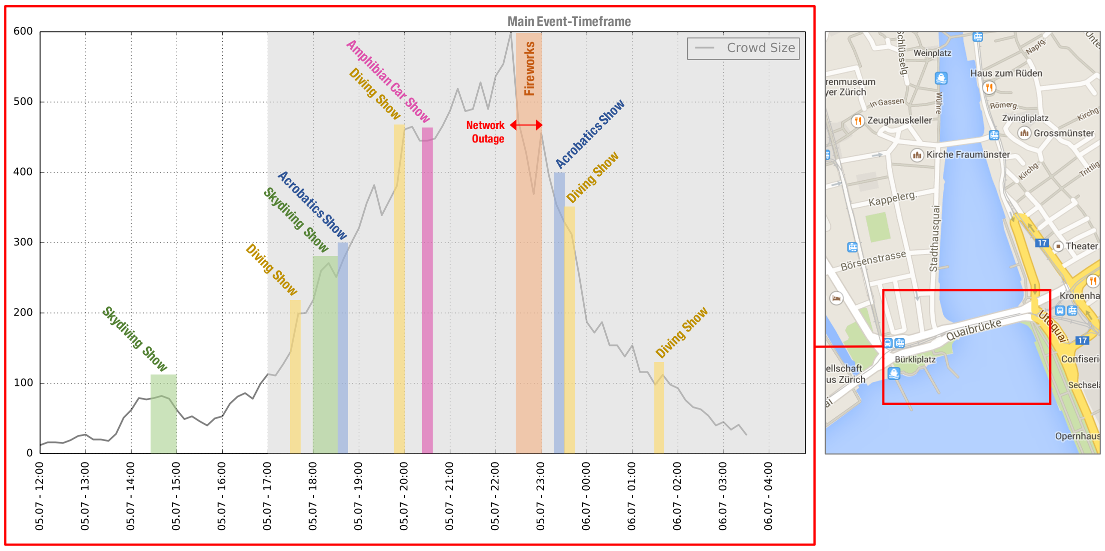
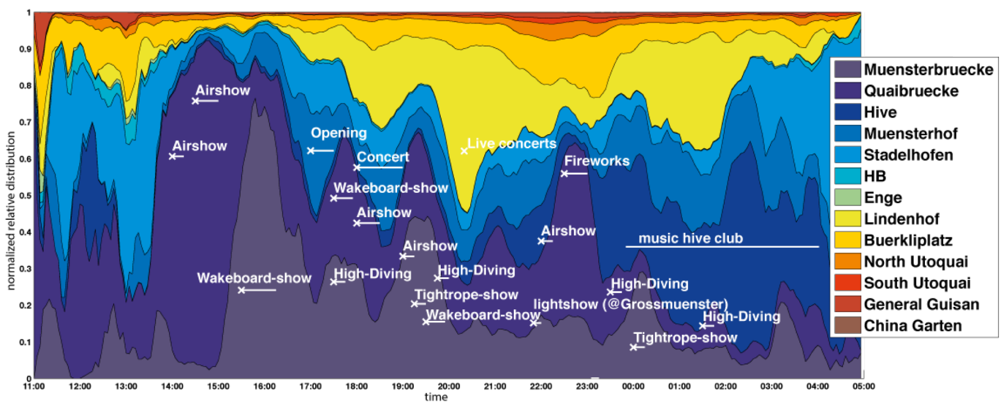


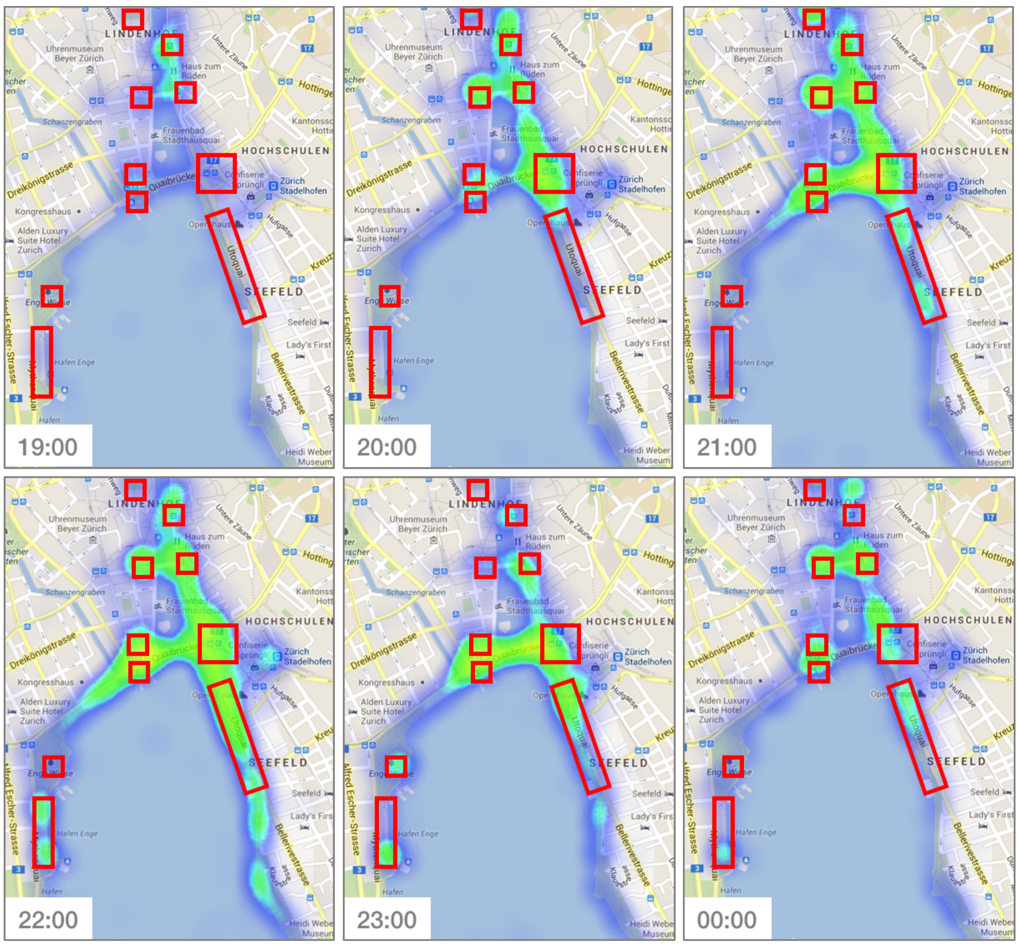
FIGURE 3.7: Crowd size analysis at one popular event location during the 2013 Zurich Festival annotated with real-life events

Also, the figure reveals one of the system's weaknesses: network outages. During the fireworks display, the large number of people gathered on the bridge overlooking Lake Zurich led to a sizable part of the crowd experiencing connectivity issues due to a network overload. While such short-term outages are not critical from a situational awareness point of view – after all, a network outage implies a very crowded area – it is problematic in terms of crowd control as the visitors in the affected area cannot be reached by means of location based messages. Therefore, chapter 5 of this thesis presents the data-driven development of a smart service that could potentially remedy this problem.

The crowd sensed data also correlated with events on a macro-scale. Figure 3.8 shows how people move between the different event locations depending on where an attraction is currently taking place. In the upper part of the figure, the starting times of attractions are marked within the location where they are taking place with a white x – their duration is indicated by a horizontal line.



(A) Relative visitor ratio between event locations



(B) Absolute crowd distribution with the main event locations highlighted

Figure 3.8: Crowd distribution over the course of a day at the 2013 Zurich Festival.

With respect to movement velocity, the Data Collection Platform also revealed some interesting aspects. Figure 3.9 visualizes the walking speeds of the visitors during three phases of the Zurich Festival. The left frame of the figure shows the proceedings during the evening of the event. It can be seen that people are stationary around bars, snack stands and event areas (see Fig. 3.8.B for details on the locations of event areas). All other areas exhibit varying degrees of motion caused by people moving between the event areas.

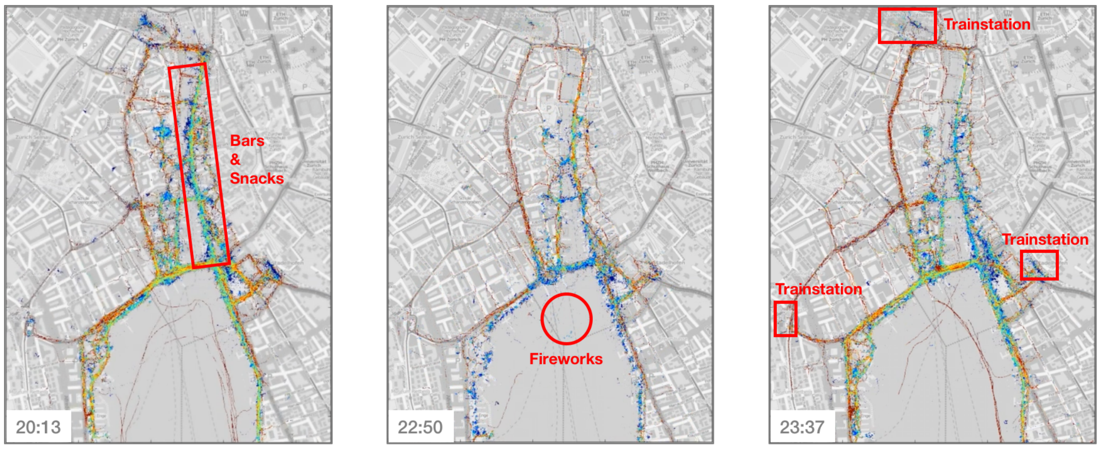


FIGURE 3.9: Walking speed distribution during different event phases – dark blue spots indicate stationary people, while dark red indicates a brisk walking pace.
(Source: adapted from [58])

The crowd behavior during the fireworks display is visualized in the middle frame of Fig. 3.9. During that time, the majority of the visitors is gathered stationary at locations with a clear view of the fireworks (especially around the shores of Lake Zurich and on bridges facing the lake). Finally, the left frame of the figure shows the time when people are starting to leave the event. Here, it can be seen that all roads leading towards the three train stations exhibit a high degree of crowd motion.

In conclusion, it can be said that during all deployments of the Data Collection Platform within the crowd management domain, its capability to provide insights into crowd behavior was considered as valuable – even in cases with a low crowd sensing app penetration, the resulting visualization of high-level crowd behavior was useful. Details about the users' perception of the platform are given in the following sub-section.

3.5.2 User Perception

With respect to user perception of the system, two separate evaluations were performed. Deployment owners and their safety and security personnel were debriefed after each deployment in form of round table discussions.

For event visitors on the other hand, a survey was created and made available via the event apps. While only a fraction of the app users participated in that survey, the total number of 722 replies was still enough to gain some meaningful insights. Table 3.4 summarizes the findings of that survey.

Gender		
Male	58.4 %	
Female	41.6 %	
Age Group		
Under 19	9.2 %	
19 – 30	33.2 %	
31 – 45	33.2 %	
46 – 60	22.4 %	
Over 60	2.1 %	
Motivation to download the app	*without mobile heat map*	*with mobile heat map*
Maps	20.9 %	10.6 %
Navigation	8.0 %	4.7 %
Messages	7.4 %	5.5 %
Social Media	3.9 %	2.4 %
Background Info	59.8 %	50.0 %
Mobile Heat Map	–	26.8 %
App Usefulness		
Very high	36.2 %	
High	46.0 %	
Low	15.4 %	
Very low	2.4 %	
Would you actively consult the app in case of an emergency?		
Yes	82.7 %	
No	17.3 %	
Would you follow the app's advice?		
Yes	94.3 %	
No	5.7 %	

When would you follow the app's advice?		
"When it comes from a known and trusted source."	49.8 %	
"If it is in line with my experience of the incident, locally."	39.8 %	
"I always trust the advice given on my smartphone"	7.1 %	
"I would not trust advice given on a phone"	1.1 %	
Other	2.2 %	
How would you communicate emergency information to others who did not have the app?		
Voice	63.1 %	
Text	35.1 %	
Other	1.8 %	
How did you find out about the app?		
Website	30.9 %	
Social Media	32.8 %	
App Store	25.0 %	
Press	11.3 %	
Did you tell your friends about the app?		
Yes	77.8 %	
No	22.2 %	

Table 3.4: Results of a user survey performed during some evaluation deployments of the Data Collection Platform (total number of participants: 722)

In general, the survey was very encouraging as more than 82 % of all participants rated the usefulness of the app as "high" or "very high". Perhaps even more encouraging is the fact that more than 94 % would follow advice given out via the app while almost 83 % would actively consult the app in case of an emergency. These findings seem to confirm the validity of the approach to use crowd sensing apps as a tool for crowd control – an observation which is also emphasized by the apps' recommendation rate of almost 78 %.

The fact that about half of the participants stated that they would only follow advice given out via an app if they knew it came from a trusted source, also underlines the importance of publishing the crowd sensing app and all its contents in the name of the deployment owner. Furthermore, it is reassuring to see that only 1 % categorically distrusts advice given on a phone.

With respect to the users' initial motivation to download the app, it became clear that the initial assumption of the available background information on an event being the driving force for app uptake, was correct. In case of apps that did not contain the mobile heat map module, almost 60 % of the survey participants claim to have initially downloaded the app in order to get access to background information on the event. In deployments where the mobile heat map module was part of the app, this number drops to 50 %. Here, it could be seen that access to the crowd density heat map was also a very strong motivation for event visitors to install the app on their phones.

During the discussions with deployment owners, the feedback was almost exclusively positive. While none of the participating authorities would consider using the crowd sensing approach as their only crowd management tool, all of them agreed that having it as an additional instrument to complement CCTV cameras and reports from on-location personnel would be very valuable. It was also stated that the demography of the event heavily influences the likelihood of the system working during an event as younger audiences are much more likely to participate. From the experts' point of view, the following pieces of information delivered by the system provided the most value:

1. What points of ingress and egress to the event were used?
2. Timing information – when did people arrive and leave?
3. How was the overall layout of the crowds during the event itself?
4. How did the crowd move during idle phases?

In summary, it was very encouraging that all participating authorities were keen on deploying the platform on a larger scale and also wanted to explore further use cases for it. Furthermore, a common feedback was that a coordinated approach including several stakeholders from both the public and private sectors would be worth exploring. In the following, a selection of quotes from the involved experts is given:

> *"The deployment of the system was the first time, the Police had knowledge about crowd movements during fireworks as CCTV doesn't work in the dark."*
>
> **Peter Clarke (City of London Police)**

"It was very interesting to see how people moved around closed roads."

Peter Clarke (City of London Police)

"The information on what happened after the event was particularly interesting."

Peter Clarke (City of London Police)

"This is one of those pieces of kit that you do not realize its true potential until you use it."

Rebecca Walker (City of London Police)

"The system is part of a toolbox and would be used in combination with reports from officers, stewards and CCTV."

Rebecca Walker (City of London Police)

"Some events like New Year's Eve are less organized – the system is especially useful in these instances as it's usually very hard to anticipate where people might move to."

Brian Drayson (British Transport Police)

"The system would also be most welcome during an evacuation."

Brian Drayson (British Transport Police)

"I am most interested in the communication perspective and the ability to communicate during an evacuation."

Alison Ingleby (Greater London Authority)

3.5.3 Influencing Soft-Factors

During the Data Collection Platform's evaluation deployments, a number of non-technical and non-scientific lessons had to be learned. In the following sub-section, the three most important are discussed.

The first soft-factor with major impact on the success of a crowd sensing deployment in the crowd management domain is the topic of marketing. It may seem obvious, but a sensible marketing strategy is essential for building a large user base which in turn is the number one requirement for successful crowd sensing applications. Ideally, working on the marketing strategy starts several months before the event as the visitors need to be aware of the event app. Ideal opportunities for app marketing are tickets, brochures, posters, websites, intro videos and of course social media channels. Ideally, visitors have downloaded the app before they arrive at the event venue in order to have a chance to receive important pieces of advice on their way.

During the evaluation deployments, the amount of effort that was put into marketing by the deployment owners varied greatly. The extreme at the lower end was the West End Live Festival – here, the entire marketing campaign consisted of a single Twitter post. Consequently, the app penetration at the event was rather poor. On the other end of the scale, the Dutch National Police had an exceptional marketing campaign for the app deployed at the 2013 Coronation Ceremony – it even included a short report that was broadcasted on TV in the scope of the national evening news. This strategy led to an extremely good app penetration: on the day of the coronation, more than 10 % of the visitors had downloaded the app.

In summary, the most successful deployments were those where elaborate marketing campaigns were launched before the event. Having said that, this doesn't necessarily mean that extreme marketing budgets are required – all that's necessary is a *smart* marketing strategy that should be executed well ahead of the event's first day. A reasonable solution is for example to print notes about the app on the official event flyers, tickets and posters and to also use social media channels frequently. None of those measures require a large budget.

The second relevant soft-factor is the degree of platform integration within the deployment owner's organization. This point also largely revolves around having a sufficient lead time before the event is taking place. It is quite vital that all the people who are in the "mission control room" during the day of the event are aware of the Data Collection Platform's capabilities. Again, this may sound like stating an obvious fact but during the evaluation deployments, it was repeatedly the case that on the day of the event, the full potential of the platform couldn't be leveraged, e.g. because the person who was responsible for visitor relations and communication was unaware of the fact that the system could send out location based messages which may also be used to provide services to the visitors. A prime example for this was the deployment at the Zurich Festival which from a data-science and situational awareness point of view was a great success. However, the safety and security forces were completely unaware of the system's communication capabilities because they haven't been briefed properly by their superiors.

The conclusion to this topic can only be that a couple of days have to be spent with training the staff that is actually responsible for managing proceedings on the day of the event. In scenarios where everybody was aware of all the platform's features and also knew how to use them, the outcome was sometimes positively surprising. The Dutch National Police for example solved a lost-child case at one of their events using the platform's messaging capability. Using the same tool, the organizers of the Lord Mayor's Show managed a number of adjustments to the event proceedings during the 2015 edition of the show which took place just one day after the terror attacks in Paris and therefore required e.g. the cancellation of the fireworks.

The third and final lesson that had to be learned during the evaluation deployments is that it is important to always include all potential stakeholders in the planning of a crowd sensing system – even if they may seem irrelevant to the actual scenario. In case of the deployment during the Dutch Coronation Ceremony in 2013, the system became the "victim of its own popularity". Due to the wide media coverage, a major Dutch telecommunications provider heard about the planned deployment and intervened at the very highest political level using the argument that the system could put such a load on its network that the carrier wasn't able to guarantee that emergency calls could be made. Naturally, this statement led to the order of not switching on the crowd sensing feature on the day of the event. Eventually, the responsible people could be convinced that the carrier's statement was in fact not true but by the time the decision was made to lift the ban on the system, the event was almost over. As a consequence of this incident, the system was only used to send out messages. This was extremely regretful as the deployment would have led to a data set of almost three times the size of the one created during the Zurich Festival.

The conclusion of this incident can only be that all stakeholders – including infrastructure providers – need to be considered during large scale deployments. Therefore, in chapter 6 of this thesis, an outlook on possible extensions of the platform, that would enable enable infrastructure providers to participate in crowd sensing deployments, is given.

3.5.4 App Usage

In section 2.5.1 it was shown how the Generic App Framework creates usage statistics. During a number of evaluation deployments, the deployment owners gave their permission to the recording of such statistics. The goal was to find out (a) if the apps' feature sets made sense and (b) how the visitors used those features. Given that the exact feature set differed between all deployments, the modules were assigned to categories which were then used as a basis for the evaluation, thereby making the deployments comparable.

Furthermore, the deployments themselves were evaluated in groups: one group contained all apps without special features, a second group contained all apps using the mobile heat map module and the third group included all apps where the gamification modules were used. The reason for this separation was that both the heat map module and the gamification modules are unique special features within the event app domain and are therefore worth investigating separately in order to find out if they added any value from a user's point of view – after all, the user survey indicated strongly, that at least the heat map module was an incentive for some people to use the app.

App Feature Category	**Usage Percentage**
Apps without special features	
Map Information	27.4 %
Background Information	30.8 %
Twitter Feeds	25.8 %
Messages	15.9 %
Apps with mobile heat map	
Mobile Heat Map	34.3 %
Map Information	14.1 %
Background Information	11.4 %
Twitter Feeds	28.1 %
Messages	12.1 %
Apps with gamification features	
Gamification Features	26.8 %
Map Information	18.8 %
Background Information	50.6 %
Messages	3.8 %

Table 3.5: App usage statistics (source: 782'058 usage logs)

Table 3.5 summarizes the app usage during the evaluation deployments based on a total of 782'058 app usage logs. In case of event apps without any special features it can be seen that the app usage is more or less evenly distributed between the basic information features: the event background information, venue layout information (consumed via a map) and social media information.

In case of apps which included the mobile heat map module, it becomes apparent that this feature has been accessed the most by a comfortable margin. This insight should encourage deployment owners to publish the crowd sensed information directly. The fact that social media information has been consumed more than geographic and background information is explained by the strategy of the respective deployment owners (the Dutch National Police) who are very active on social media. Consequently, the Twitter feeds that have been integrated into the event apps were updated frequently and were therefore interesting to the app users.

Finally, the usage analysis of apps that included gamification modules shows that these features are well received by the event visitors and therefore proofs the point that was made at the beginning of this thesis: if an app amuses people, they are likely to use it. Hence, the inclusion of gamification features can definitely help crowd sensing apps with gaining a meaningful user base. At this point it is also worth mentioning that the low usage of the messaging feature in this app group can be explained with the fact that the deployment owners during the Zurich Festival (which is part of this evaluation group) did not send any messages during the event. Consequently, the users had no reason to use the module for anything else but accessing the few messages that were published before the event.

Of course, the usage statistics don't only show how often an app feature has been used but also *when* and *where* a feature has been used and in case of map requests, which geographic location they refer to. Visualizing this information like it is done in Fig. 3.10 can therefore also help deployment owners to analyze a crowd's current state. The figure shows all app usages at 10:37 on the day of the Dutch king's coronation. At this time, the king and his wife stepped onto the balcony of their palace which is located near the center of the map.

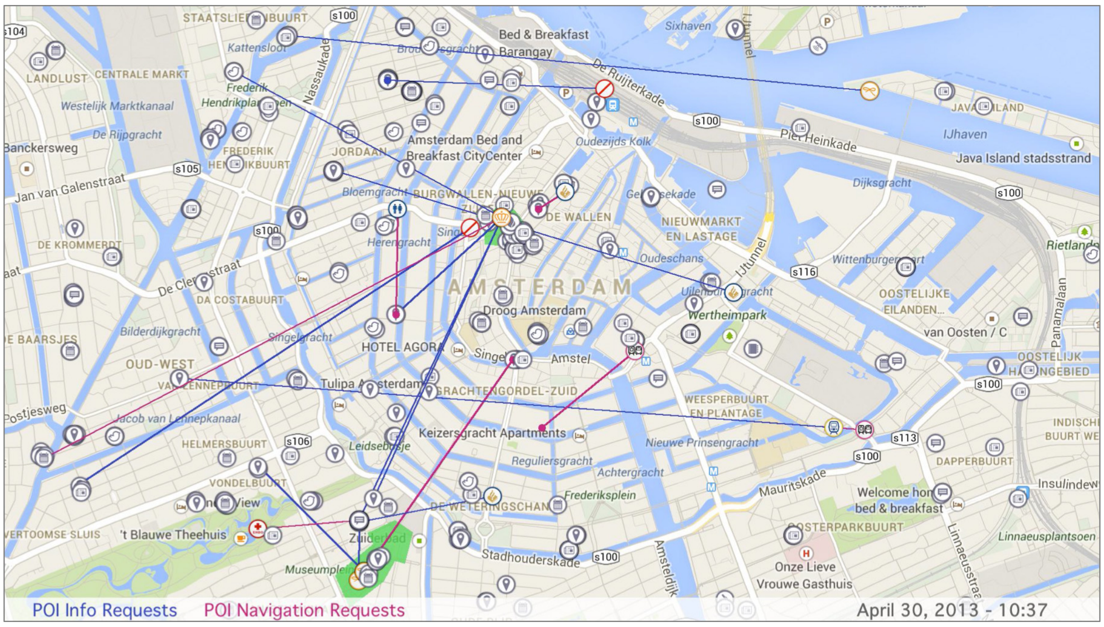


FIGURE 3.10: App usage insights during the 2013 coronation festivities in Amsterdam

The grey icons mark the locations where a feature has been used – e.g. the calendar icon marks an access of the event calendar module, the map pin icon the use of the map module, etc. The colored icons represent the locations of points of interest that have been requested by users. The location where those requests have been made is indicated by the start point of the lines ending at the respective point of interest. The color of the line represents the type of request: a blue line means that a user accessed information about the point of interest whereas a pink line symbolizes a navigation request. Visualizing app usage in this form allows insights into the users' current interests. This information in turn, can be used as an early predictor for future crowd characteristics. For example, if a meaningful part of the crowd is interested in a certain concert at an event, it can be assumed that the crowd in front of that stage is about to grow.

This relation between app usage and crowd size is also emphasized by Fig. 3.11, where the grey line indicates the number of crowd sensing app users at the Zurich Festival over its whole three-day duration. The blue line represents the usage of the event app. It can be clearly seen that changes in the app's usage curve antedate corresponding changes in the curve of the crowd size.

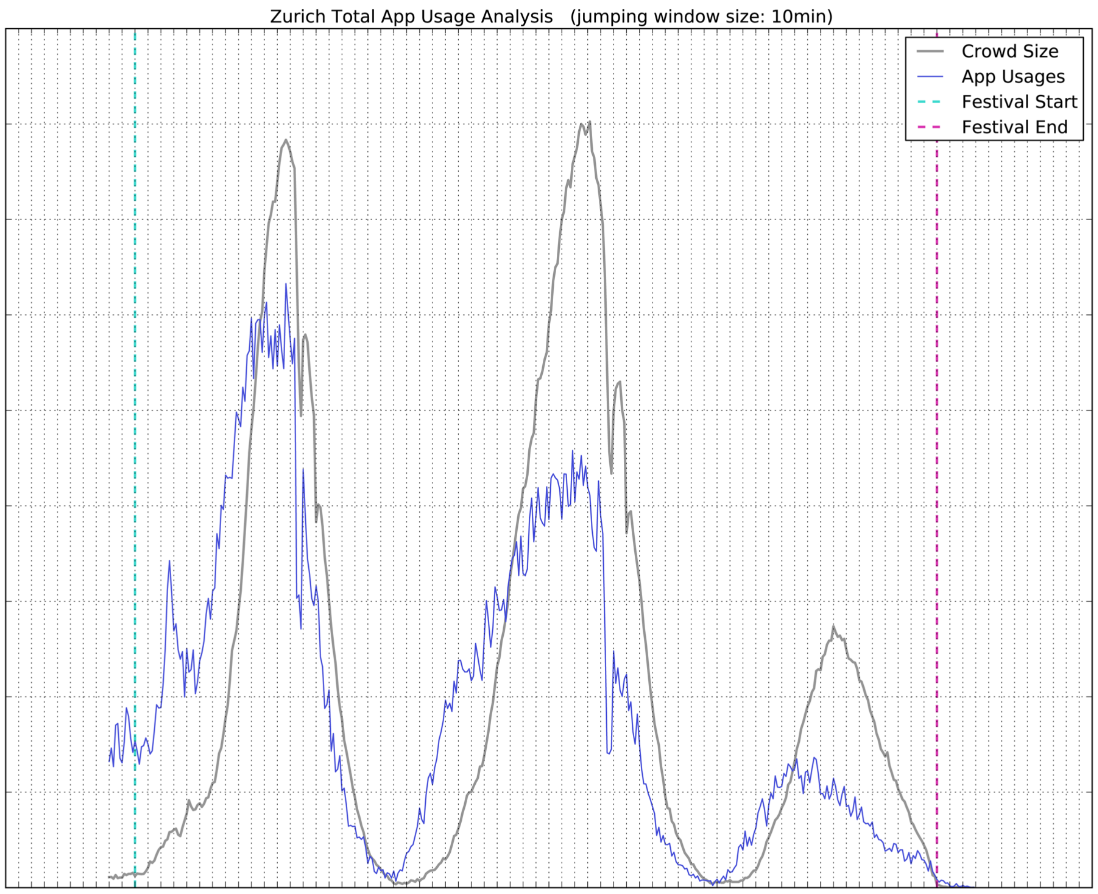


FIGURE 3.11: Correlation between the size of the crowd and the app usage during the whole three-day period of the 2013 Zurich Festival.

3.6 Chapter Summary

This chapter presented the evaluation of the proposed Data Collection Platform within in a large-scale, real-life scenario. In the introductory section, the crowd management domain was introduced and its relevance was shown. Also, the two main challenges in this scenario were explained: providing situational awareness and exercising crowd control. Afterwards, an overview about existing technologies for providing an overview about crowd behavior was given.

The second main part of this chapter elaborated on how the Data Collection Platform could be utilized in order to provide novel crowd-sensing-based tools for tackling the challenges of providing situational awareness and exercising crowd control. After the proposed approaches have been explained, their feasibility was reasoned by performing a theoretical analysis showing that it is in fact possible to estimate the behavior of a crowd based on data provided by a comparatively small number of people within the crowd. It was also shown how the ratio between the sample size (i.e. the people providing data) and the total population (i.e. the whole crowd) affects the spatial resolution within which the crowd behavior can be estimated. Furthermore, it was shown that the network communication load caused by the proposed approaches do not put an unreasonable strain on the communication network or on the users' data plans.

After the features that have been added to the platform specifically for the evaluation domain have been introduced and the deployment history has been outlined, the results of the evaluation were presented. The first part of this focused on giving an overview about the insights into crowd behavior provided by the Data Collection Platform. Here, it was shown that the system did in fact manage to provide a global and coherent overview about crowd behavior. Furthermore, it was shown that in case of deployments with a reasonable sample size, the approach also allows for observing behavior on a micro-scale – e.g. with respect to crowd movements on a bridge during different event phases.

The second part of the evaluation presented a user study on how the users of the system perceived its performance. This was done separately for users of the crowd sensing apps and for deployment owners. In case of the former the results of a survey with 722 participants was shown which indicated that the public reaction on the approach was very positive – a large majority of the participants would support the system for improving event safety. The feedback of deployment owners was gathered in form of round table discussions. Again, the reactions were encouraging with all the participating authorities having claimed that they would like to use the system on a larger scale.

In the third part of the evaluation, the three most important lessons that were learned with respect to non-technical aspects (i.e. soft-factors) were explained. In summary, it was stated that a smart marketing strategy is vital for the success of a deployment. Furthermore, it was shown that the full potential of the platform can only be leveraged with a good integration of the platform on an organizational level. The last highlighted aspect dealt with the inclusion of all possible stakeholders in the planning of a crowd sensing deployment.

The fourth and final part of the evaluation presented an analysis of a total of more than 780'000 app usage log files with the goal of giving insights into how the end users interacted with the crowd sensing apps. It was shown that the app features designed to motivate event visitors to install the apps were in fact received positively thereby further underlining the feasibility of the proposed approach. Also, it was demonstrated how app usage and crowd size development correlate with one another which led to the conclusion that app usage can be regarded as a precursor for crowd density.

In summary, this chapter showed that real-world deployments of the proposed Data Collection Platform are very feasible from both an operational and a technical point of view. It was shown that the crowd sensing system performed as expected and that all involved user groups rated the approach and its results highly.

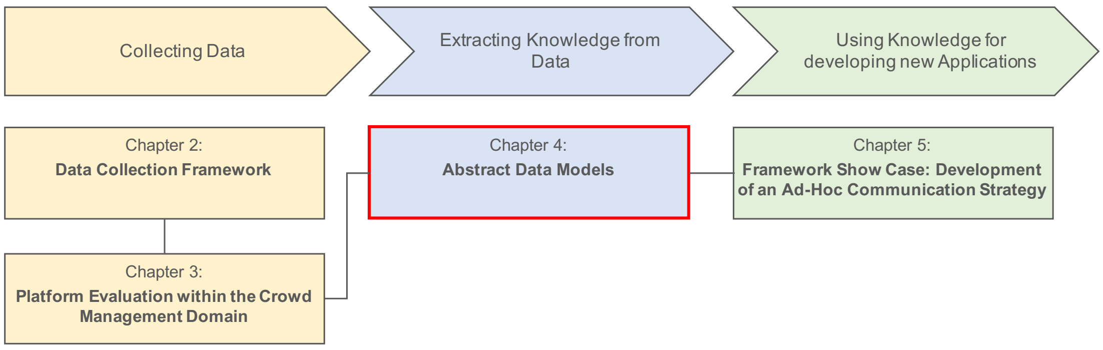
Chapter 4

Abstract Data Models

"Chaos is merely order waiting to be deciphered"

José Saramago, The Double

Given a large enough number of participants, crowd sensing data recorded in a given scenario typically represents the observed phenomena within one specific instance of that scenario very well. Based on this observation, this chapter presents a methodology for extracting abstract data models from a recorded scenario instance. Those models can then be used to e.g. simulate other scenario instances with different configurations. In addition to introducing the methodology, this chapter also presents an evaluation of the approach by revisiting the crowd management domain.



Collecting Data
Extracting Knowledge from Data
Using Knowledge for developing new Applications
Chapter 2: Data Collection Framework
Chapter 3: Platform Evaluation within the Crowd Management Domain
Chapter 4: Abstract Data Models
Chapter 5: Framework Show Case: Development of an Ad-Hoc Communication Strategy

4.1 Introduction

So far, this thesis covered the topics of collecting sensor data and using it for live or post-event analyses in its raw form. However, assuming a large enough number of crowd sensing participants, the recorded data reflects the observed phenomena within a scenario very well. Consequently, this data represents inherent “knowledge” about the particular scenario configuration. The goal of this chapter is to present a methodology to extract that knowledge in the form of abstract data models which can then be used within other applications. Specifically, it is shown how scenarios which are macroscopically similar to the one that was used for creating the data model, but which show deviations on a microscopic level, can be simulated.

This allows for experimenting with different configurations of a scenario in order to study the consequences on the observed phenomena. Usually, such simulations and evaluations require a large number of “runs” to take the effects of statistical variations in the observed phenomena into account. However, in most scenarios it is not feasible to repeat and re-record data from a scenario instance many times over. Therefore, the proposed method allows for taking the data that was recorded once and using it to drive a simulation which replicates the core characteristics of the observed phenomena with appropriate statistical variations.

Applying this concept to the domain of crowd management during large scale events means that the Data Collection Platform could not only be used for assessing crowd behavior in real-time or for performing analyses after an event is over, but that the recorded data can also be used to plan future events and to study how the crowd behavior would change if the layout of the event or the event schedule were modified – or in other words: if the event configuration was changed. By using that scenario as a show case for the methodology, this chapter provides the answer to the research question

> *How can data models be extracted from the recorded data and are they representing the inherent core characteristics of the source data correctly?*

It is demonstrated that by using the data recorded during one event day, it is possible to simulate the proceedings of another event day with a different configuration. Hence, it is not necessary to obtain another set of “training data” for that particular configuration. It is furthermore shown, that the general behavior of the simulated agents (i.e. general crowd conditions) corresponds to the real behavior expressed by people in the ground truth recording of the simulated event configuration, while at the same time realistic local behavior deviations (e.g. slightly different individual movement patterns while approaching an event location) are introduced, which leads to statistically relevant variations over the course of multiple simulation runs. In comparison to existing simulation methodologies, those deviations are not created by applying noise but by extracting realistic behavior patterns from the recorded data.

In summary, this chapter makes the following contributions to the modeling and simulation domain:

- A spatio-temporal abstraction for dividing a city into appropriate context-related cells. The abstraction is based on an in-depth analysis of the recorded data set.
- A derivation of abstract crowd motion characteristics in each cell from the data set.
- A simulation system using such a characterization to generate actual crowd motion down to the level of individual agents.
- An evaluation of the methodology in terms of being able to simulate the crowd motion/distribution of each day of a large-scale event through a model derived from the other days.

In the following section, an overview about the work that has been previously performed in the domain of pedestrian simulation is given. Afterwards, the proposed methodology for extracting abstract models from a set of crowd sensed data is explained and demonstrated at the example of a data set recorded at a large-scale, city-wide street festival. While section 4 presents the evaluation of the methodology, the final section provides a summary of this chapter.

4.2 Related Work

In the past few years, efforts have increased to derive pedestrian behavior models and to use them in simulations to measure the effect of architectural configurations on crowd behaviors [73]. With respect to simulation tools, cellular automata and agent based simulations are two very popular approaches.

On the most basic level, [74] defines cellular automata as *"examples of mathematical systems constructed from many identical components, each simple, but together capable of complex behavior."*. By analyzing such automata, scientific models can be developed and general principles applicable to complex systems can be developed. In essence, a cellular automaton is a regular lattice of cells – in the case of modeling crowd movements this is usually a two-dimensional grid. Each cell is in one of a finite number of states and the neighboring cells are able to determine what this cell's state might be. After an initial state for each cell has been defined, the subsequent changes of cell states over time are set according to a set of rules which often depend on the states of the neighboring cells.

As mentioned above, these sets of rules are usually based on a model of pedestrian behavior. One of the most well-known models is the social force model introduced by Dirk Helbing in [75]. These forces are described as a measure for internal motivations of the individuals to perform certain actions and consist of several components: the acceleration towards the desired motion velocity, the desire of an individual to keep a certain distance to other individuals or objects and a component modeling attractive effects. Taken together, they form a model that is capable of describing the self-organization of several collective effects of pedestrian behavior very realistically.

In his PhD thesis, Hubert Klüpfel presents a cellular automaton for crowd movement and egress simulation [76]. Here, a grid with rectangular cells is used to model pedestrians walking around obstacles. As a practical analysis of the proposed approach, the evacuation of passenger ships is presented. By comparing the simulation results to the observations of actual evacuation exercises, the author concludes that while there are differences in the microscopic behavior, the general macroscopic behavior of the simulation matches the reality.

In [77], Köster et al. present a cellular automaton intended for microscopic pedestrian simulations. Here, a hexagonal cell grid is used allowing the "pedestrians" more degrees of movement freedom. The resulting model has been used to simulate the evacuation of a football stadium and the crowd movements through a train station. Apart from the cells' shape, this model is quite similar to the one presented in [78]. Here, Burstedde et al. have introduced the concept of the floor field which was shown to be an efficient way to simulate pedestrian behavior in a way that also allows for observing phenomena like lane formation.

Zia et al. also perform an evacuation simulation based on a cellular automaton with the goal of studying the effects that "leaders" in a crowd may have on the general crowd behavior [79]. Here, leaders are defined as persons who are assisted by a piece of ambient intelligence giving them insights into the most suitable exit for an evacuation. Furthermore, a social dimension is added to the model as the concept of trust is introduced – a leader has more trust than other simulated persons and is therefore able to cause parts of the crowd to also choose the most efficient exit (which may not be equal to the closest exit). The work demonstrates how even a small portion of leaders in a crowd leads to a remarkable difference in the useage of exits that would otherwise be under-utilized.

As mentioned previously, agent based simulations are an alternative to cellular automata. This approach describes a system that is modeled as a set of autonomous decision making entities called agents. Each agent individually assesses its situation and makes decisions on the basis of a set of rules [80]. In contrast to cellular automata, an agent based simulation is not discrete with respect to its notion of time and space and can therefore produce more realistic results – on the downside however, they are more complex and need more computing resources.

[81] presents an early agent based simulation for evacuation scenarios which also incorporates the concept of leadership. It is shown that the results of this simulation are very similar to the observations made during a controlled experiment. However, the authors also state that more input for the rules governing the agents' behavior is required in order for the model to work on a large scale. In [82] Johansson et al. present an agent based simulation of extreme crowding situations. The authors also conclude that on a microscopic level it is very difficult to calibrate the model because of the level of detail and the lack of available empirical data.

The need to run pedestrian simulations on a large scale is emphasized by Zia et al. who demonstrate in [83] that the results from a small scale simulation may considerably deviate from those of large scale simulations due to the dynamics of large crowds. To enable efficient large scale simulations, the authors introduce the concept of evidence based simulations (run in an agent based fashion), where *"empirical evidence is collected at the microscopic level, to inform a model on the macroscopic level. The validity of the large scale model is supported by model parameters coming from empirical data, but also from model-reality feedback loops at runtime."*. In a different study, the authors also employ agent-based simulations to study the effects of togetherness and dispersion in sub-groups of a crowd [84].

The simulation engine used for running the model developed in the course of this chapter is also following an agent based approach. In fact, the simulation framework presented by my colleague Andreas Poxrucker in [85] and [86] has been used for this work as it was specifically designed to run simulations in urban environments and offers a great deal of flexibility with respect to model integration. Originally, the simulator has been developed for researching multi-modal mobility phenomena in smart cities.

In his book [87], Michael Batty presents a comprehensive overview about simulations in the city-domain utilizing both cellular automata and agent based simulations. These models are applied at several scales ranging from the street level to patterns of an entire urban region, thereby underlining the strengths and weaknesses of both simulation approaches. Again, it is stressed that apart from choosing the right simulation tool, a very critical aspect is how to actually create the model or in other words: how to "train" the simulation.

In order to calibrate the model parameters (i.e. the training process), experiments under controlled conditions can be performed and used as reference data – an approach that has been followed in [32] and [88]. This also includes analyzing video footage showing pedestrian dynamics as it was done in [66]. An interesting alternative is presented in [89] where the authors present a BDI agent based simulation (BDI: belief, desire, intent) for emergency response. Here, the agents are trained in a virtual reality environment.

The common ground of all the above simulation approaches (and many others) is that they either require a large effort for calibrating the model or that they work with global models such as the social force model [75] by using e.g. differential equations. The agent-based approach presented in this chapter however, learns local behavior deviations that may be caused by characteristics of the specific current situation and location (e.g. during an ongoing fireworks display on a bridge) based on a set of training data and then formulates behavior rules that agents in the affected areas express during the respective times.

At this stage, it is important to highlight the difference between the aims of this research and existing work on pedestrian dynamics simulation. The majority of the later assumes that for each agent the intended destination is known, at least on a statistical level. Such simulators then use some sort of model of individual motion (again, possibly on a statistical level) and interactions between people to derive the overall crowd dynamics in a constrained space. By contrast, this work shows how to derive the distribution of destinations/walking directions of people in different classes of constrained urban spaces from a real-life human mobility data set based on the role of the respective space within a city-wide event. Thus, this approach must be seen as complementary to classical pedestrian dynamics simulations allowing such simulations to initialize the destination distribution for each sub-space of the city from real life data and be combined into a realistic city-wide simulation for a particular type of event.

4.3 Methodology

The following section presents the methodology for extracting abstract data models from crowd sensed pedestrian movement data during large scale events. The first sub-section gives a quick overview about the process. In the second sub-section, each step is explained in detail at the example of a data set recorded during a multi-day, city-wide street festival.

4.3.1 Overview

Before looking at the actual methodology for creating data models based on crowd sensed data, it is important to list the assumptions that were made with respect to the nature of the scenario as they provide a clear idea about the nature of the scenarios this methodology can be applied to. Specifically, the proposed approach is based on three assumptions:

1. It is assumed that a large scale event consists of a number of spatially and temporally constrained and distributed sub-events or attractions such as concerts, snack stands, etc.
2. It is assumed that the aforementioned sub-events and attractions are a driving factor behind most of the crowd motion and distribution. In other words, most of the people attending the event, do so to visit one or more of the sub-events or attractions.
3. It is assumed that the way crowds move through and behave at different locations is determined by the abstract type of location (e.g. street, open space, train station, etc.) and the relation of this location to the sites where sub-events or attractions are taking place. In other words, people will behave differently on a street leading towards a concert than they do while being on an open space in front of a concert stage.

If those assumptions are met by the scenario, the proposed methodology can be applied. Figure 4.1 provides an overview of the model extraction process which takes place in five separate phases.

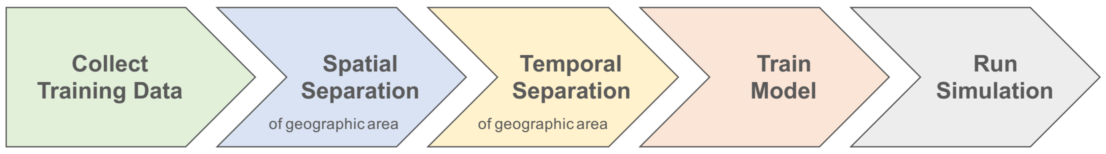


FIGURE 4.1: Overview about the process of creating abstract data models based on movement data that has been recorded during large scale events

The very obvious first phase of the process is the actual collection of training data during a crowd sensing scenario. Here, it is vital to gather as much data as possible throughout all the different stages of the event in order to collect the maximum possible amount of reference data for all details of the event proceedings.

During the second phase, the geographic area where the scenario is taking place is separated into regions representing areas of similar crowd behavior. Examples include street crossings, event locations, train stations, etc. These regions represent one important input parameter for the model.

The third phase of the process is a temporal separation of the geographic area. Here, the idea is to add temporal semantic information to the regions identified in the previous step – in case of regions where concerts are taking place for example, the times during which those concerts are happening are recorded.

In the fourth step, the actual model is created. This is being done based on the data that has been recorded in each of the different types of region during each of the different temporal phases. The result of this step is a micro-model for each possible combination of the two separations of the geographical area performed in the previous phases. For example, one of those micro-models could describe how people were moving while being on a street leading towards a major attraction just before the attraction started. Furthermore, a macro-model is created, describing the rough distribution of data over each type of geographical region. This model is used for example to determine the locations where people are arriving from when entering the event premises.

The fifth and final stage is performing the actual simulation. Here, the data models are fed into a simulation engine which is then started with a certain configuration of the scenario. In case of large scale events, such a configuration consists of the event layout (e.g. the location of attractions) and the event schedule (e.g. when attractions are taking place).

4.3.2 In-Depth Description

In the following, the detailed description of the proposed methodology for extracting abstract data models is given at the example of a data set created during one of the real-world deployments performed during the evaluation of the Data Collection Platform.

Phase 1: Collecting Training Data

The step of gathering training data for the model in the scope of a crowd sensing deployment lays the foundation for the following steps. Here, it is vital to record enough data so that all the phenomena characterizing the scenario are accurately represented in the data. Applying this to the crowd management domain at hand means that data needs to be gathered during each stage of the event and also at all relevant locations.

For the scope of this work, the data set created during the 2013 Zurich Festival [58] which was also mentioned in the previous chapter was used. In total, this data set consists of approximately 25 million data points that have been contributed by roughly

28’000 people over the total duration of the three-day festival. At the time of the writing of this thesis – and to my best knowledge – this data set was the largest set of its kind. Each data point consists of a timestamp, a position value and speed and heading information.

Using this data set as a basis, a total of 35 hours were selected as training data, thereby covering all event phases during each of the three days. Out of those 35 hours, the first and the third event day made up 11 hours each, while the second – and longest – event day spanned over a duration of 13 hours.

Phase 2: Spatial Separation

In order to be able to determine local behavior patterns, it is necessary to separate the geographical area where the scenario is taking place into distinct sub-regions each with a specific semantic meaning. In the scope of this paper, these sub-regions are referred to as cells. Normally, the shape and size of those cells doesn’t matter and is completely scenario-dependent.

In the smart city domain in general and in case of the crowd management scenario in particular, it seems obvious that the actual city infrastructure should be taken as a basis for the spatial separation into cells. Therefore, a so-called *street graph* is created which approximates a city’s infrastructure. Figure 4.2 shows an example of such a street graph.

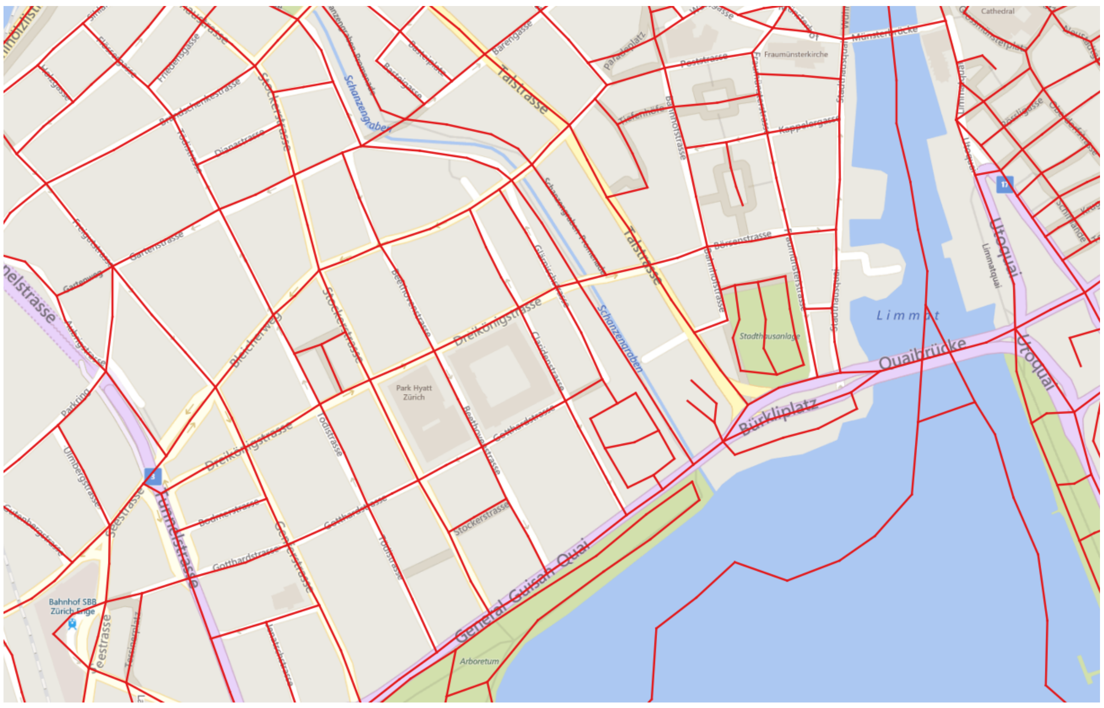


FIGURE 4.2: Detail view of a street graph

While a few details are simplified and minor infrastructure items such as blind alleys are omitted, it can be seen that the street graph represents the city's infrastructure pretty well. The graph also includes some elements that are not directly associated with actual streets. Namely, those are extra edges surrounding and filling open spaces (like parks) and "border edges" located in water expanses. The logic for those extra elements and the simplification of the street layout shall become clearer in the next step of the spatial separation. In order to get an overview about the result of the first step of the spatial separation, Fig. 4.3 shows the complete street graph of the Zurich Festival area.

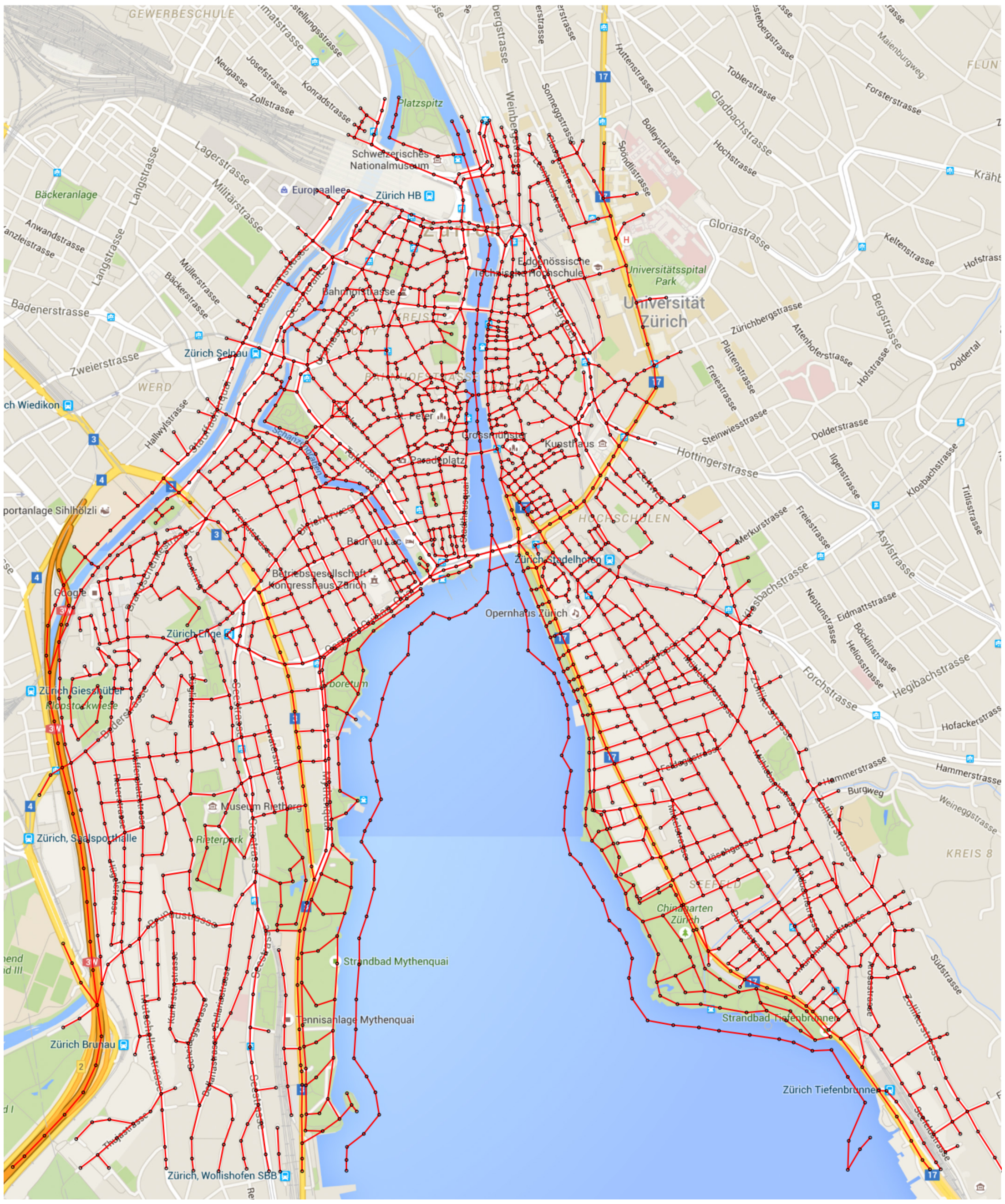
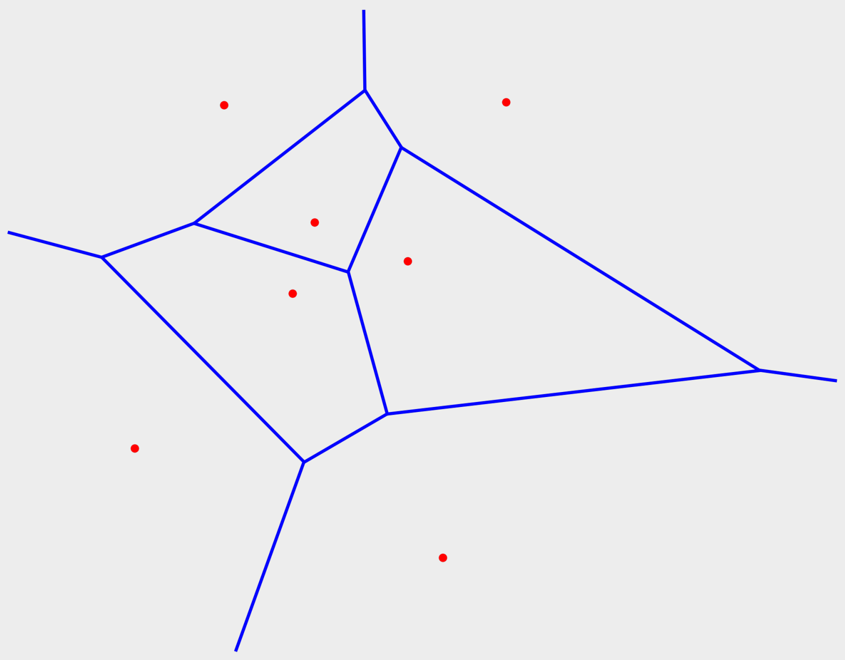


FIGURE 4.3: Complete street graph for the event area of the 2013 Zurich Festival

The second step of the spatial separation is to create the actual cells based on the street graph. For this, the concept of a Voronoi diagram needs to be introduced.

Definition: Voronoi Diagram

Given some number of points in the plane, their Voronoi diagram divides the plane according to the nearest-neighbor rule, whereas each point is associated with the region of the plane closest to it. (Source: [90])



Computing the Voronoi diagram of the street graph leads to a rather natural separation of the event area for the simple reason that Voronoi diagrams are created based on the nodes of a graph. In a street graph, those nodes are typically placed at locations where people are likely to express a certain behavior – for example at road junctions, a change of movement direction is likely, whereas on straight sections of a street, such a change is much more unlikely. Consequently, the resulting cells are created around those important graph nodes representing areas with distinct behavior patterns.

At this point, it also becomes clear why extra elements have been inserted into the street graph. The graph edges inserted around the shores of Lake Zurich have been added in order to create cells which only contain water and can therefore be disregarded for the computation of the movement model. The extra edges that have been inserted into open areas (e.g. parks) have the purpose of dividing those open spaces into several cells therefore allowing to identify local behavior patterns within those open spaces with a finer granularity.

Figure 4.4 shows the spatial separation of the geographic area into distinct cells based on the Voronoi diagram of the street graph. Irrelevant cells – i.e. cells outside of the event area or cells containing nothing but water – have been removed.

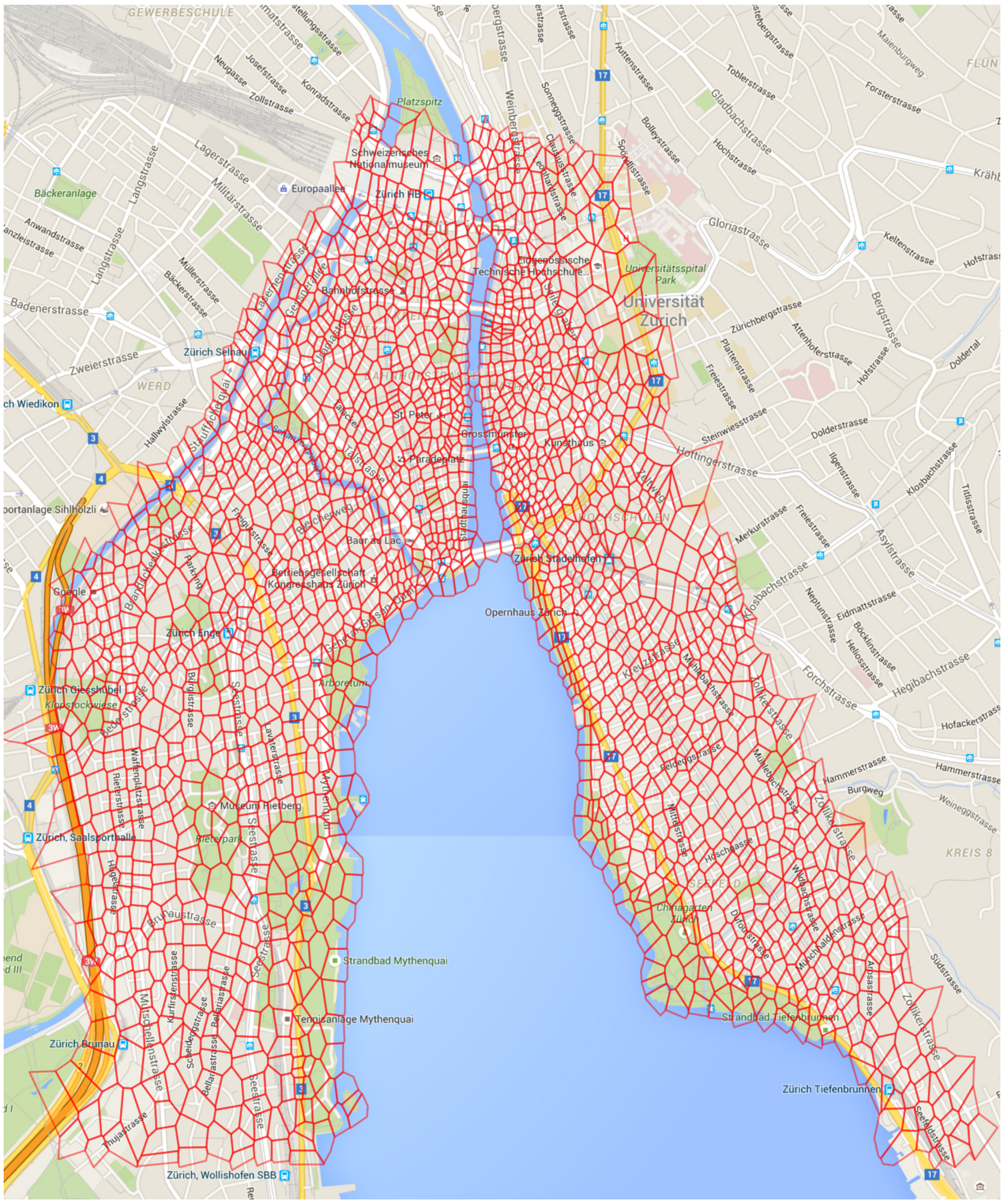


Figure 4.4: Spatial separation for the event area of the 2013 Zurich Festival

The last step of partitioning the geographic area into cells consists of assigning semantic meaning to each cell. For that, the following twelve cell types were introduced:

1. Event locations

2. Event streets
3. Train stations
4. Event-related open spaces
5. Open spaces
6. Main streets
7. Side streets
8. Residential areas
9. Crossings between main roads
10. Crossings between a main and a side road
11. Crossings between side roads
12. Crossings with event streets

Assigning those types to all cells leads to a complete spatial and semantic separation of the geographic area – the basis for being able to identify areas of local behavior patterns. Figure 4.5 shows the result of this process.

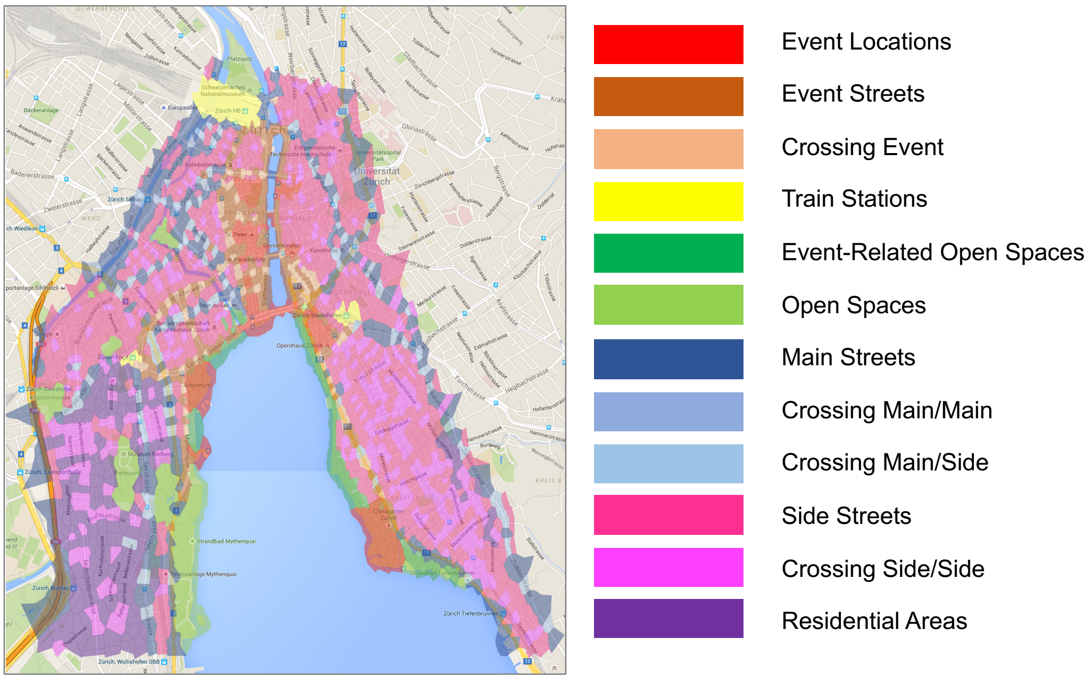


FIGURE 4.5: Spatial and semantic separation for the event area of the 2013 Zurich Festival

Determining Boundary Conditions

Before continuing with the next phase, it is important to determine the boundary conditions of the model. This requires identifying those cells that contain enough data points to contribute to a solid data basis for the model – i.e. finding those cells where enough training data is available to achieve decent results.

Figure 4.6 shows a visualization color coding each cell according to the amount of data points that are located inside the respective cell. It can be clearly seen that the regions where the Zurich Festival was actually taking place as well as the main streets connecting those locations, contain the majority of the data points. Regions that are further away from those areas, are generally sparsely populated with respect to the number of data points. Also, there are some regions where no training data is available at all. Consequently, the model will not be as precise in those regions compared to the actual event area.

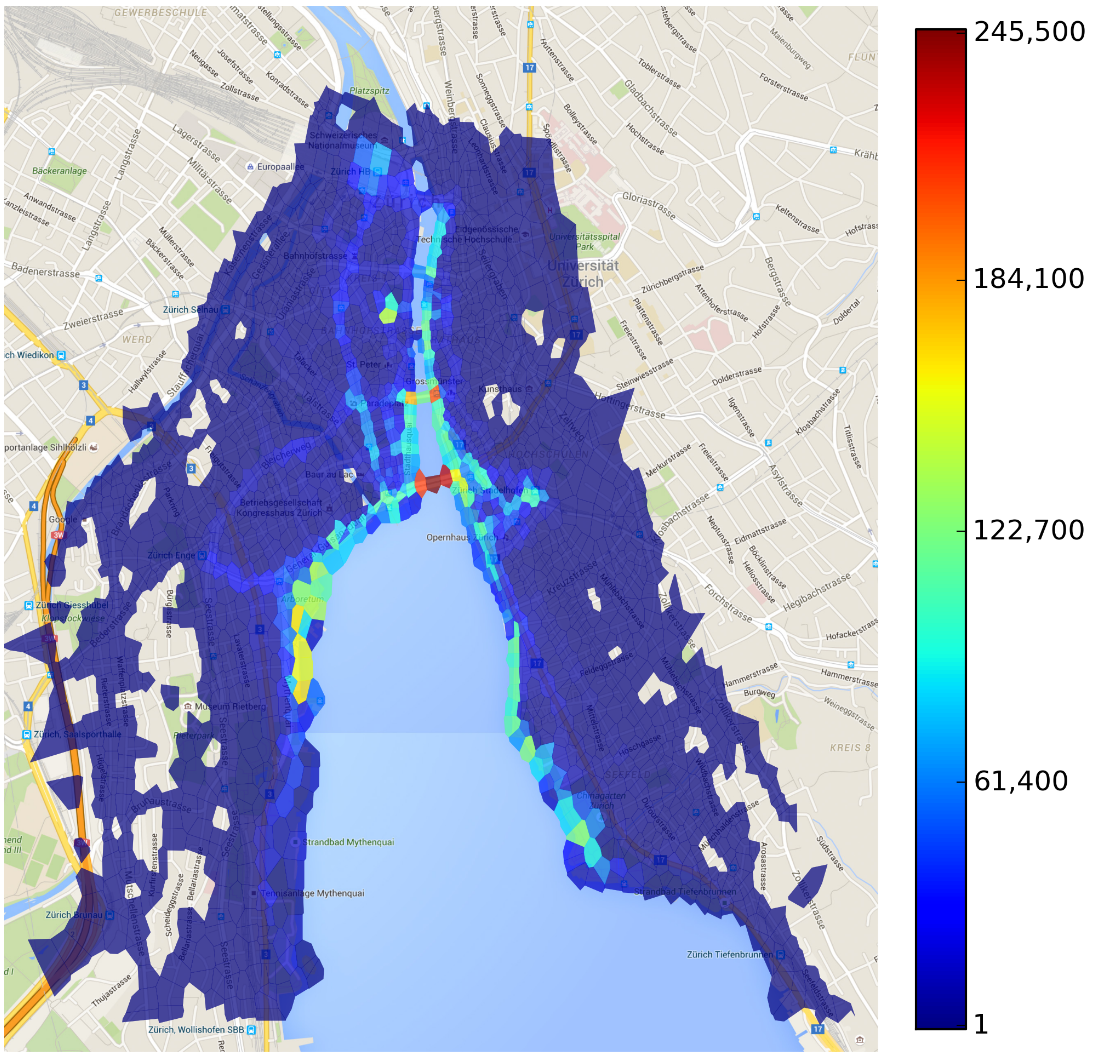


FIGURE 4.6: Number of available data points in the training data per cell

Phase 3: Temporal Separation

The purpose of the third phase of the model extraction process is to record the timing information of the scenario. This happens on two levels. Firstly, the overall scenario is divided into distinct temporal stages which are part of a given scenario configuration. They can also be thought of as the scenario's "macro schedule". In case of the Zurich Festival example, the following stages were defined:

1. Arrival
2. Event stage 1 – i.e. people moving between different attractions and concerts
3. Pre-fireworks – i.e. people gathering at suitable locations for watching the fireworks
4. Fireworks
5. Post-fireworks – i.e. people leaving the locations where they observed the fireworks
6. Event stage 2 – i.e. people moving between different attractions and concerts
7. Departure

Secondly, the scenario's "micro schedule" is recorded. In case of large scale events this is the actual event schedule which contains information about when an attraction is taking place at which location. In other words, this schedule defines, when the areas that have been labels as "event locations" in the previous phase are "active" – i.e. when they are of relevance to the event visitors.

At the end of the temporal separation phase, all the required information for building the model has been computed: the training data has been made available, the geographic area has been partitioned into meaningful sub-areas and the semantics of the scenario (i.e. the information on the event proceedings and their timing) have been recorded.

Phase 4: Training the Model

The actual model to be created for simulating crowd movements during a given event configuration of the Zurich Festival consists of two parts:

1. The macroscopic part defining the ratio of people in each cell type during each stage of the event.
2. The microscopic part defining the actual movement of people in each cell at each stage of the event.

As the macroscopic part of the movement model only defines the approximate distribution of people over the event area, it can be computed in a rather simple fashion by iterating in one-hour time frames over the full training data within the defined event stages while computing the ratio of people within each cell type. Based on this, the average development of the crowd's distribution within each region of the event area is computed. That level of granularity is enough to gain knowledge about the main regions of ingress and egress of people as well as to identify basic high-level crowd shifts throughout the day.

The macroscopic model will be used to both define the initial distribution of simulated agents and to identify regions and approximate time frames when new agents need to be spawned or removed from the simulation for each of the event phases. An example for this could be the ingress of people via the city's train stations during the major arrival periods.

Before the microscopic part of the movement model can be computed, some pre-processing needs to be done. Specifically, for each cell of the event area the following features are calculated in 10-minute jumping window time frames over the full training data within the defined event stages:

1. The distribution of people's heading directions relative to each street within the cell.
2. The distribution of people's movement speeds.
3. The distribution of people's heading directions relative to each event location that is marked as "active" during the particular time frame.
4. The distribution of people's heading directions relative to each train station.

The purpose of this pre-processing step is to make the semantics of the event part of the model. Instead of purely calculating e.g. the general movement direction, the crowd movements in relation to the entities, that actually make up the event, are calculated. Therefore, an agent in the model isn't merely moving into direction x with a certain likelihood but the agent is moving e.g. *towards a concert* which happens to take place during a certain time at a certain location according to the event's configuration.

With respect to the computation of movement speed and heading values it must be said that those numbers were calculated purely based on the available location information as the speed and heading values calculated by the smartphones' operating systems themselves turned out to be distorted in some instances. The most prominent example being the heading direction that is calculated by the smartphone operating system partly based on the data of the device's built-in compass which tends to deliver faulty data when for example walking past power generators which are sometimes deployed at large scale events.

After this pre-processing step, the microscopic model can be created. Here, the actual behavior of the agents is defined by three parameters:

1. Movement speed
2. Heading direction
3. The time interval, the current behavior is valid for

Those parameters are computed by analyzing the pre-processed training data within each cell of each cell type during each event stage. The result of this analysis is a probabilistic function for each parameter which is depending on the event stage and the cell (type). These functions are then evaluated during the simulation in order to assign movement patterns to simulated agents. The details of this process which is also illustrated in Fig. 4.7 are given in the description of the next phase.

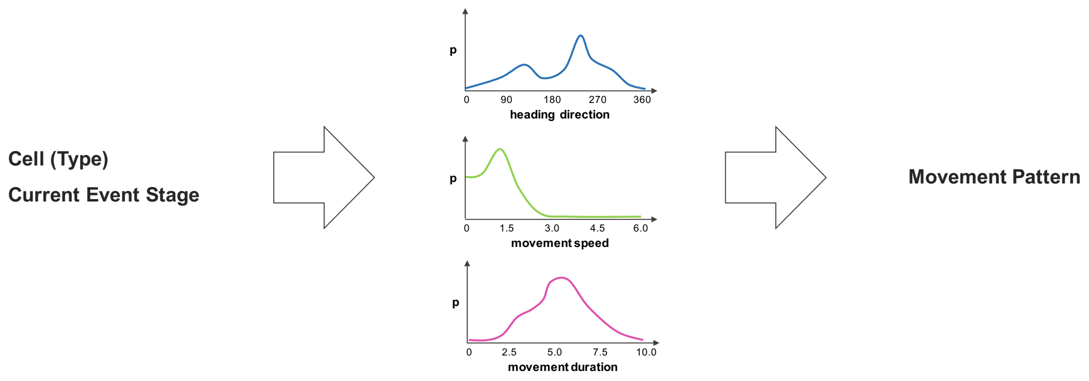


FIGURE 4.7: Illustration of movement pattern computation

Phase 5: Running the Simulation

As mentioned previously, the simulation engine used for this work was developed by my colleague Andreas Poxrucker for the purpose of researching multi-modal mobility in smart city environments [85, 86]. This simulation framework uses the time-discrete, multi-agent simulation environment NetLogo [91] as an engine driving the execution of the simulation. The movement model itself as well as state and behavior models of agents were implemented as a NetLogo Java extension essentially providing a “move" procedure to be called from NetLogo once per time step for every agent.

During the initialization of the simulation, the basic spatial boundaries (i.e. a bounding rectangle) of the simulated environment are determined by constructing a street graph of the area of the Zurich Festival from a shapefile. Furthermore, the cell and cell type information are read from the event specification. Hence, in this step, the configuration of the event is defined. The temporal bounds (i.e. the starting and ending date and time for each event phase and the event in general) are also set according to the event specification. Additionally, an initial population of agents is created whose number can be adjusted. The initial agents are placed randomly on the map with the total distribution defined by the macroscopic part of the movement model. Also, an initial set of behavior parameters is assigned to each agent based on the microscopic part of the movement model.

During the simulation, each simulation step is associated with a configurable time interval, e.g. 1 second. During every turn, the “move" procedure of each agent is called executing the following workflow: First, the agent decides whether it needs to update its movement parameters by checking the time interval, the current behavior parameters are valid for. If an update is required, the movement model is invoked to deliver new parameters. Therefore, the agent first determines the current event stage based on the current simulation time as well as the cell it’s currently located in. Using the current event stage and cell as input for the movement model, the agent is then assigned a new direction, speed, and time interval based on the probabilistic functions delivered by the microscopic model for each of the parameters. After this update step, the agent moves on the map using a function of the current location, movement direction and speed to determine its new location.

It is important to note that the movement of the agents is only guided by the parameters provided by the movement model. It is not restricted by the edges of the street graph loaded from the shapefile. The reason for this is that the movement parameters do not necessarily align with the direction of streets. For example, agents may move across squares, open spaces or on pavements next to the actual street – for none of which a representation is given in the street graph which makes them hard to describe in an efficient manner. Consequently, the agents can move freely on the simulation plane and are only stopped by the shores of Lake Zurich or the basic spatial boundaries of the simulation environment. If an agent steps out of the latter, it is removed from the simulation.

At the end of each simulation step, the macroscopic part of the movement model is evaluated. If it dictates a higher population than the existing one, new agents are spawned in those areas defined by the macroscopic model (e.g. in train stations). Those new agents are initialized in the same manner as mentioned above.

4.4 Evaluation

The following section evaluates the quality of the abstract data model extracted from the crowd sensing data recorded at the 2013 Zurich Festival. This is being done in two parts: first, the feasibility of the identified behavior patterns is illustrated at the example of a concrete event area. Afterwards, the correlation between the model and the reality is discussed.

4.4.1 Feasibility of Identified Behavior Patterns

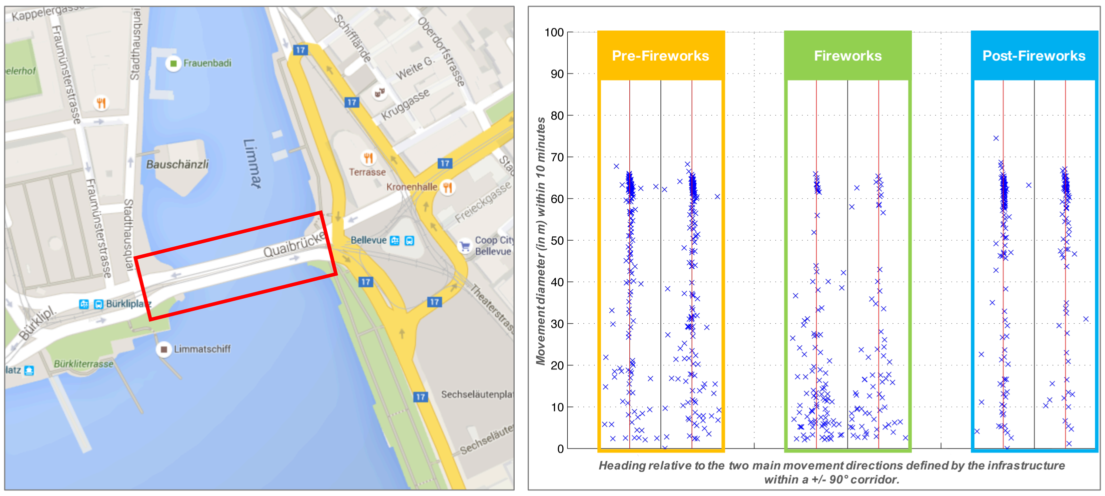


FIGURE 4.8: Insights into behavior patterns (right) observed on the "Quai Bridge" (left) before, during and after the fireworks at the Zurich festival

The purpose of this sub-section is to provide a general feeling for the feasibility of the identified behavior patterns (and therefore the data parametrization) by looking at a particular part of the event area during a particular time of the event proceedings in detail. For that, the "Quai Bridge" (see the left image in Fig. 4.8), which runs along the North shore of Lake Zurich, was chosen. This choice was made partly because it connects the two main regions of the event area and partly because it is one of the best locations to observe the fireworks display – the attraction that was chose for this analysis (which also happens to be one of the main attractions of the whole festival).

Infrastructure-wise there are two main directions of movement possible on the bridge: in Eastern and in Western direction. The graph on the right hand side of Fig. 4.8 shows the identified behavior patterns before, during and after the fireworks. The x-axis denotes people's heading direction relative to the two main directions of movement (which are marked with red lines) within a +/- 90 ° corridor. The y-axis denotes people's movement radius in meters per 10 minutes. Very low values can be assumed as more or less stationary, given the potential minimal GPS inaccuracies and the observation that people standing in crowds at festivals still tend to move a bit when letting others pass by. The graphs clearly show how the majority of people is moving along the main directions of movement at a reasonable pace (given the crowded environment) before and after the fireworks display. There are only a few who are facing either side of the bridge while being stationary or who are walking to either side of the bridge. During the fireworks however, the behavior changes drastically: a much larger part of the crowd is stationary or moving very slowly while at the same time facing the sides of the bridge where the fireworks are on display.

This example plainly shows that the behavior patterns extracted from the data by the proposed approach are feasible with respect to real-world events. That evidence is further highlighted by visually comparing the recorded ground truth data with one frame of a simulation run as depicted in Fig. 4.9. Therefore, it seems fair to make the preliminary conclusion that the model seems to generally produce realistic crowd movements.

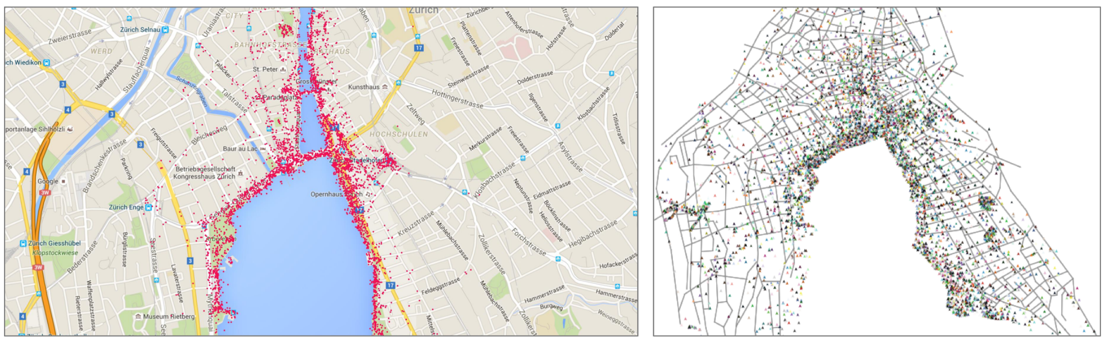


FIGURE 4.9: Comparison between ground truth data (left) and simulator output (right) during the fireworks at the Zurich festival

4.4.2 Correlation between Model and Reality

In order to evaluate, how well the model performs, the Zurich data set was separated into sub-sets for training and testing. Out of the three available festival days, two were taken for training the model, while the configuration (i.e. timing and locations of attractions) of the remaining third day was used for a simulation. For the sake of getting meaningful comparison outcomes, the simulation was performed 100 times each. The result of every simulation run was then compared to the ground truth data of the third day.

With respect to the comparison itself, an evaluation-grid was laid on top of the festival area. The bounding rectangle of this grid was constructed using the street graph's extreme points resulting in a bounding box of 3'430 m width and 3'622 m height, which was separated into grid cells measuring approximately 100 m in each dimension (leading to a total of 1'224 cells). The percentage of people in the current cell of both the ground truth and the simulated data were then mapped to this grid in jumping windows of 15 minutes length. In other words, the the ratio of the overall crowd within each cell within that 15 minute time window was calculated for both the simulation and the ground truth.

In order to determine how well the simulations match the actual, recorded (ground truth) behavior, for every time window, a t-test was performed for each of the grid cells, comparing the average density and standard deviation calculated from the 100 simulation runs to the average density of the ground truth data. An alpha value of 0.05 was chosen which resulted in a significance of 95 %. Furthermore, the average power of the test was fairly high ranging between 91 % and 93 %. For each time window, this resulted in a partitioning of the grid cells into cells with reasonably similar expected density on the one hand and dissimilar cells on the other.

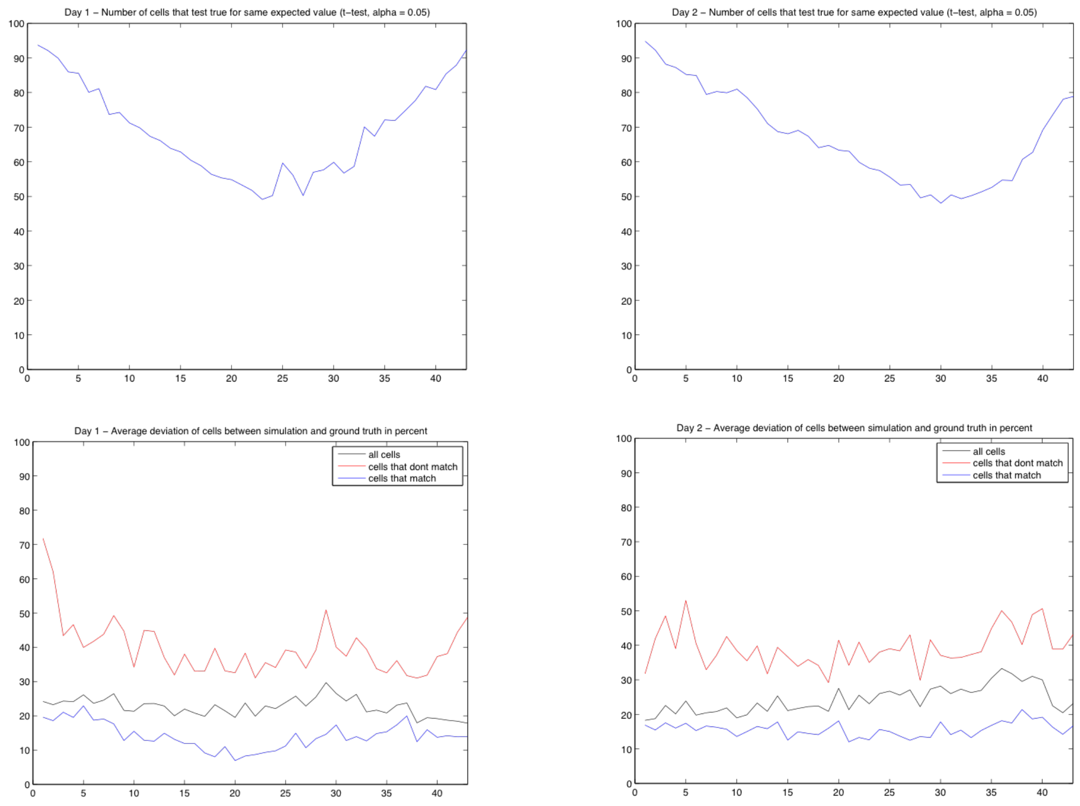


FIGURE 4.10: Evaluation of the data model. Top: the percentage of cells where simulated behavior matched the ground truth for two test days. Bottom: the average deviation (in %) between simulated density and ground truth density for two test days.

The top graphs in Fig. 4.10 show the percentage of similar cells out of all cells for each time window for two test days. It can be seen that towards the peak of the festival day (i.e. the time with the most visitors), the percentage of cells where simulated and ground truth behavior match, reaches a low point. The main reason for this is to be found in the fact that a large number of cells actually doesn't provide enough training data in order to form correct behavior rules for those cells (see sub-section "Determining Boundary Conditions" for details). Hence, the more virtual agents walk into those "ill-defined" areas, the more relevant will the deviation in simulated vs. ground truth behavior be. The important fact however is, that the behavior within the well-defined cells (i.e. the region where the festival actually takes place) matches quite well.

As further metric to gauge the quality of the model, the average deviation of simulated density vs. real density (in percent) was calculated for similar and dissimilar cells as well as for all the cells (see the bottom graphs in Fig. 4.10). It can be seen that in case of the dissimilar cells, the average deviation is about 40 %. This value might seem high at first sight, but it needs to be taken into consideration that the value, this deviation refers to, is itself a percent value describing the ratio of the overall festival crowd in that particular cell. Hence, it might be that in reality, about 0.05% of all event visitors were in a given cell (a realistic value for the sparsely populated grid cells given the large number of evaluation grid cells), whereas the simulation placed 0.07% of the virtual agents in that cell. For areas that are not genuinely important to the actual event proceedings, this seems like an acceptable deviation.

A further explanation for deviations can be found in the fact that the simulated agents are not bound in their movements by the physical infrastructure (see the sub-section "Running the Simulation" for the reasoning). Hence, it could be that some agents may take shortcuts that don't actually exist in the real world which in turn may lead to some deviations in the long run.

4.5 Chapter Summary

This chapter presented a methodology for extracting abstract data models from a set of crowd sensed data and demonstrated the process at the example of the 2013 Zurich Festival data set. In essence, it was therefore shown that crowd sensing systems can provide value beyond the live or post-event analysis aspects.

In the course of developing this methodology, three main contributions have been made. Firstly, a spatio-temporal abstraction for dividing a city into into appropriate context-related cells was presented. Here, it was shown how computing the Voronoi diagram based on the city's extended street graph leads to a separation of the city's infrastructure into cells which are suitable for studying local behavior patterns as they are created around areas where such patterns are likely to emerge. Subsequently, a set of types or categories was assigned to each cell in order to include the scenario's semantics in the geographic abstraction. At this point of the process it is possible to identify for example all areas where attractions were taking place. Afterwards, temporal semantics (e.g. start and end times of attractions) was added to the cells thereby completing the configuration of an event.

The second main contribution was the actual derivation of abstract crowd motion characteristics for each of the cells based on a set of training data. Here, it was shown how the data is first pre-processed in order to be able to analyze crowd movements in relation to the scenario's semantics (e.g. movements relative to the location of an attraction) as opposed to only being able to analyze general movement directions. In the second step, it was shown how the model is built based on this pre-processed data. The results of this step are a macroscopic part of the model defining both the overall distribution of the crowd over the entire geographical area (and the major points of ingress and egress) and a microscopic part of the model defining the actual movements on an agent-level based on the current stage of the event and the cell (type) within which an agent is currently located. After the model was created, it was demonstrated how it is applied to an existing simulation framework.

The chapter's third main contribution was an evaluation of the methodology. It was shown how it is possible to train a model with data from two event days and simulate the proceedings of the third day with a suitable accuracy. In the course of this discussion it was also shown how regions with a smaller amount of training data affect the quality of the overall model given that the model will be less precise in those areas which leads to overall deviations in the crowd behavior. On a qualitative level, it was shown that the identified behavior patterns are in fact feasible and match the expected behavior in the real world.

In summary, it can be said that the proposed methodology does indeed produce realistic crowd motions within a simulation, thereby underlining the hypothesis that crowd sensed data contains inherent "knowledge" about a scenario which can provide value beyond the live or post-event analysis aspect. In practice, this means that simulations of different configurations of a scenario can be performed without the need to record reference data for each new configuration.

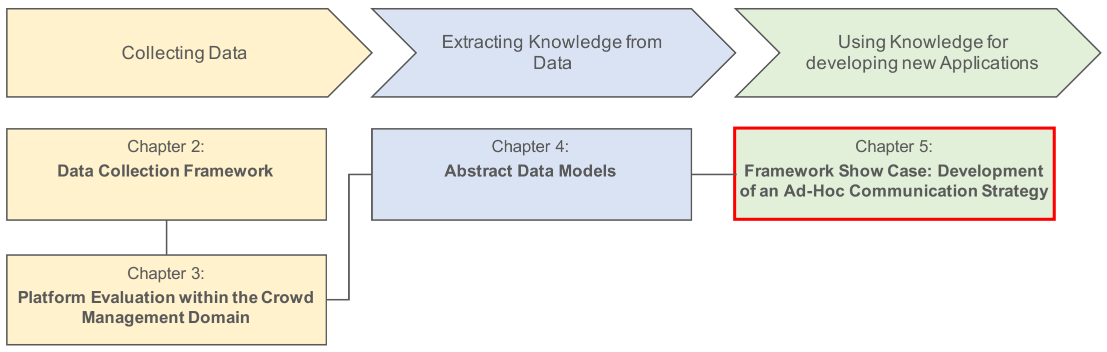
Chapter 5

Framework Show Case: Development of an Ad-Hoc Communication Strategy

"The single biggest problem in communication is the illusion that it has taken place."

George Bernard Shaw

After having covered the data recording and modeling aspects in the previous chapters, chapter 5 presents the development of a Smart City service based on the data model resulting from those preceding steps. This is being done at the example of a mobility-based Ad-Hoc communication approach to be used for sending advice and information to people in case of partial network failures. Therefore, the relevance of this thesis is underlined by presenting a practical application of the proposed concepts.


Collecting Data
Extracting Knowledge from Data
Using Knowledge for developing new Applications
Chapter 2: Data Collection Framework
Chapter 3: Platform Evaluation within the Crowd Management Domain
Chapter 4: Abstract Data Models
Chapter 5: Framework Show Case: Development of an Ad-Hoc Communication Strategy

5.1 Introduction

In the previous chapters it was discussed how crowd sensing data can be recorded in large-scale, real-world scenarios and how the inherent "scenario knowledge" that is contained within the data can be extracted from it by means of abstract data models. These models have been shown to be suitable for running rather realistic simulations which in turn can be used to evaluate alternative configurations of the simulated scenario.

In this chapter, it is shown how the abstract data models can also be used for developing new applications in a data-driven fashion. The reasoning behind this approach is that very complex scenarios have a tendency of showing non-deterministic effects which cannot be predicted in advance. Instead, these effects are only observable – either in real life or in the scope of a simulation. Clearly, observations in the real world are very expensive both with respect to time and money. Therefore, the approach of running a large number of simulations of a scenario in parallel and studying the occurring effects is a lot more sensible – provided of course, that the simulation is (a) accurate enough and (b) ensures a certain statistically relevant variance between each run of the simulation. Given that the last chapter has shown that the proposed modeling approach results in simulations satisfying both of these criteria, this chapter will present how an actual application can be developed based on such simulations.

As the crowd management domain has been used as a practical application for all the topics discussed in this thesis so far, it shall also be used as the show case scenario for this chapter. One of the potential problems of using a smartphone-based approach for exercising crowd control during large scale events is to be found in the fact that smartphones generally depend on some sort of communication infrastructure. If this infrastructure breaks down – which tends to happen from time to time, especially in case of extremely crowded scenarios – it means that there is no way to send messages containing advice or warning to the visitors. Potentially, this could have severe implications considering that those extreme conditions are the ones where crowd managers are most likely to send advice to people on the premises.

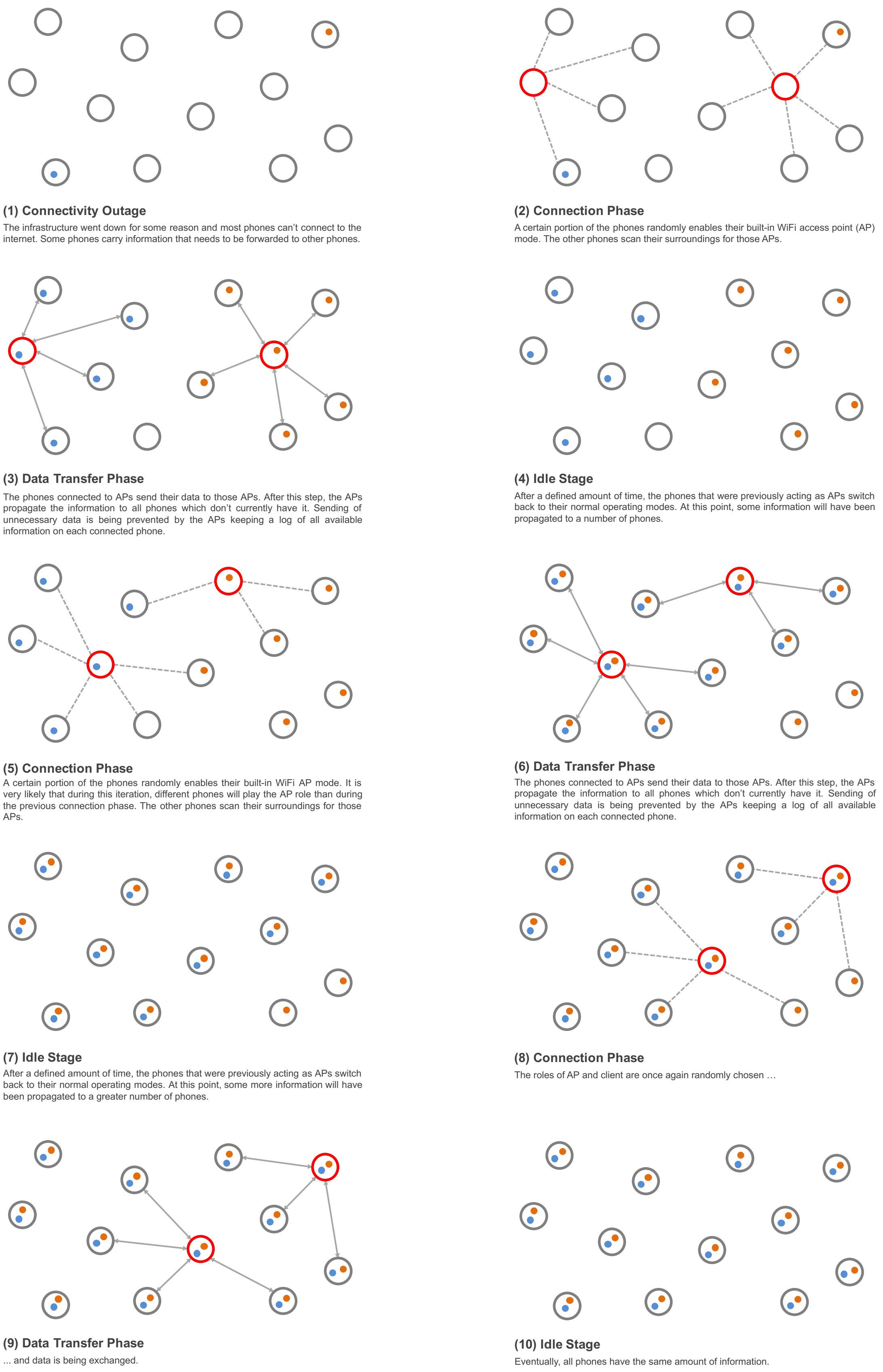
For exactly these situations, the research community has developed a number of Ad-Hoc and peer-to-peer (P2P) approaches enabling devices to communicate directly with one another without the need for a centralized communication infrastructure. In the crowd control scenario, the theory would be that as long as one single device is able to receive a message, it can forward it to other devices which in turn continue forwarding the message until all devices in the crowd received the message. Unfortunately however, most existing Ad-Hoc approaches cannot be run on off-the-shelf smartphones as they don't allow third party apps to access the necessary hardware features. While standards have been developed to solve this problem, those standards are still not available to most devices on the market. Section 5.2 will provide an overview about the Ad-Hoc domain in general and the problems related to smartphones, in particular.

However, one Ad-Hoc approach seems to be promising in terms of compatibility with the current smartphone generation: opportunistic networking. The idea of this concept is that a device only needs to be able to (a) connect to other devices via standard WiFi and (b) to create a WiFi hotspot on its own. Both these criteria are met by the vast majority of smartphones currently available on the market.

The process of opportunistic networking is illustrated in Fig. 5.1. If a connectivity outage occurs in an area, a fraction of the affected devices is creating a WiFi hotspot. The other devices are trying to connect to one of the available hotspots. This way, a number of "micro networks" is built – each capable of synchronizing all participating devices. After a certain time, the micro networks are dissolved again. At this point, the cycle is re-started by a fraction of the devices creating a WiFi hotspot which leads to the formation of micro networks – here, it is important to design the process in a way that it becomes unlikely for the micro networks to have the same topology as in the previous cycle. Statistically speaking, it is ensured that after a certain amount of cycles all devices will have synchronized.

It is obvious that the factor defining the performance of this approach is the mode switching strategy – i.e. the logic of (a) assigning the roles of client and server to the devices and (b) of making the decision which client connects to which server. In general, the current opportunistic networking approaches are either applying a purely random mode switching strategy or they try to optimize the strategy based on a set of global parameters (again, please see section 5.2 for details). In the scope of this work, a novel mode switching strategy is proposed which is based on the mobility of the devices – a key feature of the crowd control scenario. The parameters for this strategy are optimized based on a large number of simulations of the scenario using the movement model created in the previous chapter.

Previous works have presented smartphones as a simple "tool" to enable opportunistic networking without actually determining what conditions have to be met by the overall system in order for the smartphones to be utilized to their full capacity from a network performance point of view. Therefore, this chapter furthermore provides the results of an empirical analysis of the capabilities and limitations of a current set of smartphones, with respect to opportunistic networking. These findings are also considered for the formulation of the proposed mode switching strategy and are part of the model parameters used within the simulation.



(1) Connectivity Outage
The infrastructure went down for some reason and most phones can't connect to the internet. Some phones carry information that needs to be forwarded to other phones.

(2) Connection Phase
A certain portion of the phones randomly enables their built-in WiFi access point (AP) mode. The other phones scan their surroundings for those APs.

(3) Data Transfer Phase
The phones connected to APs send their data to those APs. After this step, the APs propagate the information to all phones which don't currently have it. Sending of unnecessary data is being prevented by the APs keeping a log of all available information on each connected phone.

(4) Idle Stage
After a defined amount of time, the phones that were previously acting as APs switch back to their normal operating modes. At this point, some information will have been propagated to a number of phones.

(5) Connection Phase
A certain portion of the phones randomly enables their built-in WiFi AP mode. It is very likely that during this iteration, different phones will play the AP role than during the previous connection phase. The other phones scan their surroundings for those APs.

(6) Data Transfer Phase
The phones connected to APs send their data to those APs. After this step, the APs propagate the information to all phones which don't currently have it. Sending of unnecessary data is being prevented by the APs keeping a log of all available information on each connected phone.

(7) Idle Stage
After a defined amount of time, the phones that were previously acting as APs switch back to their normal operating modes. At this point, some more information will have been propagated to a greater number of phones.

(8) Connection Phase
The roles of AP and client are once again randomly chosen …

(9) Data Transfer Phase
… and data is being exchanged.

(10) Idle Stage
Eventually, all phones have the same amount of information.

FIGURE 5.1: Illustration of opportunistic networking

It is important, to point out that this work does not claim to have developed of a completely new Ad-Hoc approach. Instead, the point is to demonstrate that the quality of existing Ad-Hoc approaches can be increased in certain scenarios by means of running data-driven simulations for the parameter optimization. Therefore, this chapter makes the following concrete contributions to the field of opportunistic networking:

- An empirical analysis with respect to the boundary conditions imposed by smartphones on the performance of an Ad-Hoc networking approach.
- The formulation of a novel mode switching strategy for opportunistic networking based on node mobility. The parameters for this strategy are optimized using a simulation based on the crowd movement model presented in the previous chapter.
- An evaluation of the mobility-based mode switching strategy based on a large-scale simulation which was shown to exhibit realistic crowd behavior. Here, it is shown that the proposed mode switching strategy performs better than traditional random approaches.

The following related work section provides an overview about the field of Ad-Hoc networking in general and highlights the challenges with respect to mobile Ad-Hoc within large crowds in particular. Afterwards, the boundary conditions imposed by smartphones on the performance of Ad-Hoc networking are presented in the form of an empirical analysis. Section 4 will then present the proposed mobility based mode switching strategy. The results of the parameter optimization are also shown in this section. Before a summary of the chapter is given in section 6, the penultimate section evaluates the proposed mode switching strategy and also compares it to a traditional, purely random approach.

5.2 Related Work

The task of establishing communication networks in the absence of a fixed infrastructure (i.e. building ad-hoc networks) has been researched for decades and was initially motivated by military and disaster recovery motives. Especially the field of mobile Ad-Hoc networks (MANETs) has gained increased interest in the 1990s [92].

Traditional Ad-Hoc approaches work on the TCP/IP stack and require its modification on almost every layer. Most modern smartphones (e.g. iPhone and Android devices) on the other hand, do not grant access for developers to those parts of the operating system to make the necessary modifications. However, given that an app-based scenario requires a system that can be used on standard off-the-shelf hardware, it is no option to pursue an approach which calls for a rooted device. To solve this problem, the WiFi Alliance introduced the WiFi-Direct standard which facilitates direct communication between WiFi-enabled devices. Unfortunately, the vast majority of smartphones currently available on the market does not support this standard [93], so it is not feasible in the scope of this work.

[94] gives an overview about the challenges facing the Ad-Hoc networking research community when it comes to disaster relief and introduces the Twimight Twitter client which has a built-in disaster mode activated when the smartphone loses network connectivity. Tweets are then propagated between adjacent devices via Bluetooth. Given the cumbersome pairing process and the limited range of Bluetooth, this solution also seems to be unfeasible for the proposed scenario.

Trifunovic et al. introduced WiFi-Opp [95] which lays the basis for the work presented in this chapter and is designed as a delay tolerant opportunistic network. WiFi-Opp relies on the fact that modern smartphones have a built-in WiFi-hotspot functionality. The underlying principle is that if a certain amount of devices take over the role as access points and the others are trying to connect to those, information can be exchanged between the connected devices. If those modes are switched periodically, the information is propagated to all participating devices. To assign modes to devices, WiFi-Opp employs a random strategy which is further refined in [96]. [95] also shows that WiFi-Opp is more energy efficient than the WiFi-Direct standard.

Since the scenario examined for the development of this chapter's mobility-based AdHoc networking approach usually involves large (and mostly dense) crowds of people, the effect of those crowds on the WiFi signal needs to be considered. These signals are mainly transmitted in the 2.4 GHz range – which happens to be more or less exactly the resonance frequency of water [97]. Incidentally, the human body is made up of up to 72% water [98]. Therefore, people must have an effect on the transmission of WiFi signals – a fact that is even more relevant in the given scenario consisting of potentially very dense crowds. [99] establishes that attenuation due to the human body on the WiFi signal strength is stronger when a person is closer to an access point than when it is further away. Fet et al. quantify these findings in [100]: when a person is standing 1m away from an access point facing the device, the RSSI of the WiFi signal drops over 40 dBm when receiving the signal behind that person compared to receiving it in front of it. In a distance of 10m however, there is no noticeable difference anymore. In practice, this means that smartphones which are carried in a pocket by their users are "weak" access points and one cannot assume that they will cover the same distance that can be observed during line-of-sight experiments. This observation has been factored into the proposed mobility-based AdHoc networking strategy.

Of course, a number of publications discusses approaches to make mobile AdHoc networks more efficient. [101] for example considers network traffic and usage of different WiFi-channels as part of the mode switching strategy which leads to a quicker and more energy efficient spreading of information – an approach which cannot be pursued in the scope of regular apps as the operating systems' APIs don't allow access to the required level. In [102] SPONET is introduced which implements a context sensitive mode switching strategy by learning the user's movement behavior to identify communication opportunities at frequently visited locations. Due to this learning phase of movement patterns, the approach is not ideal for the rather spontaneous scenario discussed in the scope of this work. The E-DARWIN system [103] introduces a fixed schedule-based switching strategy which was designed for scenarios without device mobility – and can therefore also not be applied in the given context.

[104] introduces an adaptive mode switching strategy which is geared towards optimizing battery drain from a global perspective. It deals with the fact that devices acting as WiFi-hotspots have a higher power consumption than devices acting as clients. The work aims at minimizing the time that each device needs to spend as WiFi-hotspot by estimating the remaining contact duration as a function of the elapsed contact duration. While this approach has not been implemented in the current version of the proposed system, it seems well worth to further investigate.

This chapter is largely based on [105] where our research group first introduced both the concept of using a model created from crowd sourced data for developing a complex system and the mobility-based mode switching strategy. In that work however, the model used for the development process was a lot more limited than the one presented in this thesis which led to a less optimized mode switching strategy.

At this point, it should be mentioned that none of the related work has been implemented on the Apple iOS platform. The work presented in this chapter is no exception as it focuses on Android, the mobile operating system which has currently the largest market share. The reason for focusing on Android is to be found in the iOS API which restricts access to vital system level functionalities. Having said that, the approach will of course also work on iOS devices if Apple should decide to open up the necessary APIs one day.

Finally, one point from the introduction shall be emphasized: this chapter's proposed mobility-based AdHoc networking approach is not to be understood as a novel way of performing AdHoc networking. Instead, this work's main contribution is the optimization of the parameter set of the well-established concept of opportunistic networking with the help of a data model extracted from real-world data with the goal of improving the performance of the existing concept.

5.3 Boundary Conditions Imposed by Smartphones

Existing opportunistic networking approaches do not consider the special characteristics of smartphones in their mode switching strategies and are therefore formulated for general purpose computing nodes. However, in order to design a mode switching strategy and a communication protocol which are optimal for the crowd control scenario, an understanding about the capabilities and limitations of smartphones from an opportunistic networking perspective is needed.

Given that to my best knowledge no such research has been carried out so far, a series of experiments was performed with a whole host of smartphones. The set of devices was assembled based on the most popular devices manufactured between 2010 and 2014, thereby providing a good overview about the market at the time of the experiment. In the following, the outcome of this empirical evaluation is presented.

(A) Device Type

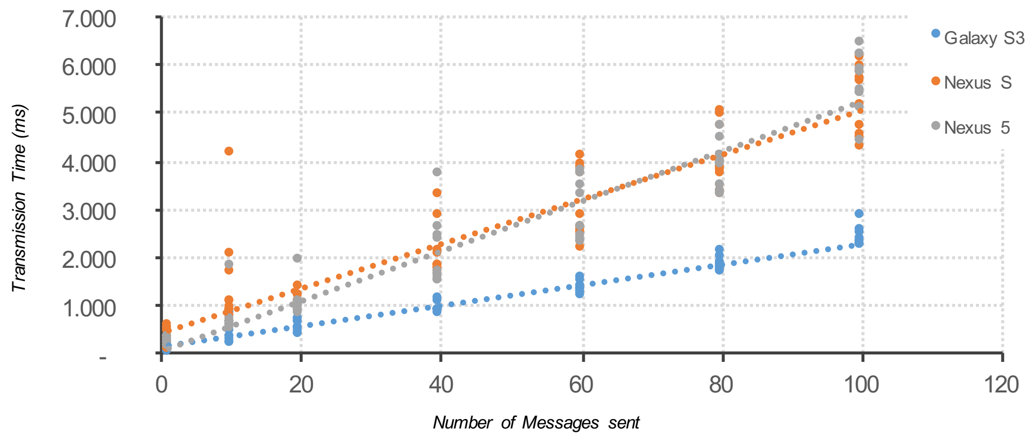


Figure 5.2: Message transmission times for different smartphone types

To learn if older smartphones performed worse with respect to opportunistic networking than more modern devices, a series of tests was run where a fixed number of messages (each 100 characters long) was transmitted between an access point device and a client device. As reference devices, the Google Nexus S (released 12/2010), the Samsung Galaxy S3 (released 05/2012) and the Google Nexus 5 (released 10/2013) were chosen. The findings clearly showed that the oldest device performed broadly on the same level as the newest device, with the "middle aged" device surprisingly outperforming both (for details, please refer to Fig. 5.2). Therefore, it can be concluded that the device type does not have to be regarded as an important factor for either the mode switching strategy or the communication protocol.

(B) Payload Size

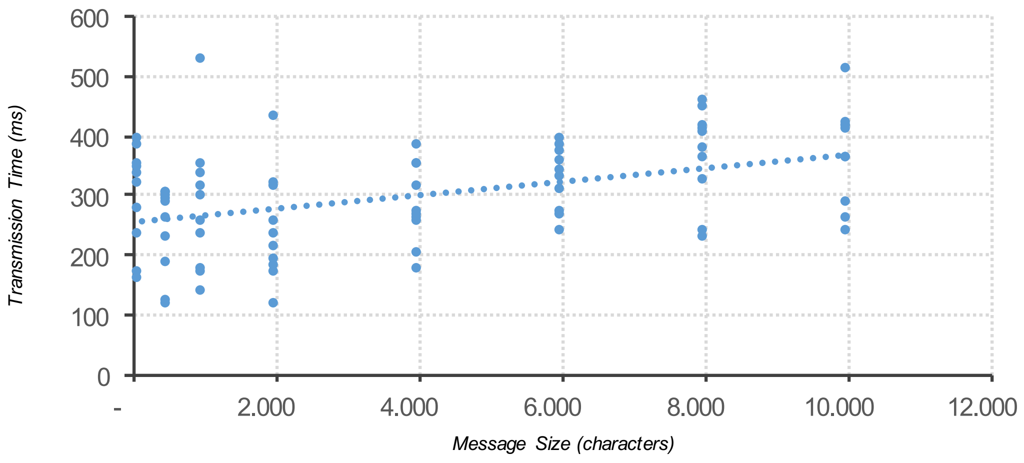


Figure 5.3: Message transmission times for different payload sizes

In order to find out what impact the payload size has on the transmission speed, a series of tests was performed where messages of varying size were exchanged between two devices with payloads ranging between 100 and 10'000 characters. The outcome of those experiments showed that while there seems to be a linear dependency between the message size and the transmission time, it is very small and therefore does not have any meaningful impact on the communication protocol. Figure 5.3 demonstrates that the average transmission time only grows by 32 % when the message size is increased by a factor of five (∼280 ms vs. ∼370 ms).

(C) Packaging Strategy

One key design decision for the communication protocol was whether to exchange all messages within one large packet or if each message should be transmitted as a single small packet. Consequently, a series of experiments was performed, where the packaging strategy was altered for each iteration. The first iteration consisted of a single 100'000 character packet being sent while the last iteration consisted of 100 packages each 1'000 characters long.

The experiments showed that the former performed about 7 times better than the latter (for details, please see Fig. 5.4). This effect seems to be caused by the protocol overhead which is considerably greater when a large number of packets is sent and cannot be offset by the increase in pure transmission time which is relatively small as it was shown in Fig. 5.3. As a result of this experiment, the decision was made to exchange messages between devices in bulks.

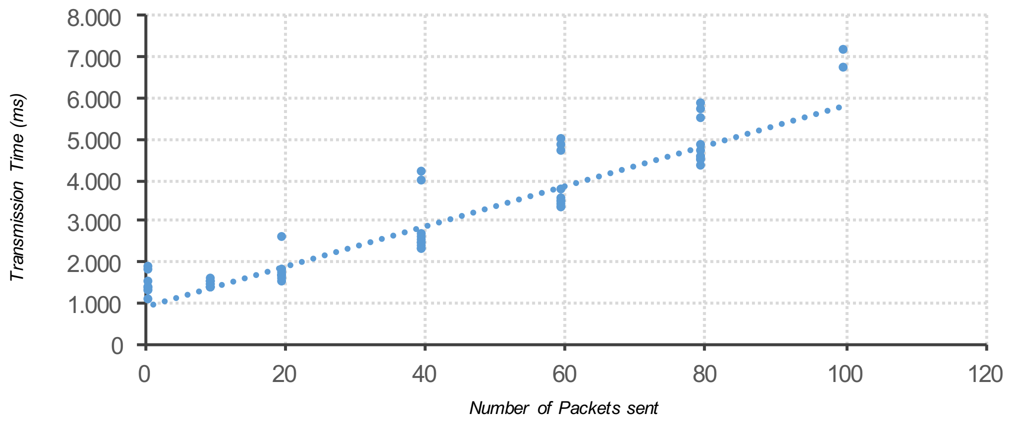


FIGURE 5.4: Message transmission times for different packaging strategies

(D) Distance between Devices

A further experiment was performed to find out if the distance between two devices had a large impact on transmission times – especially in scenarios with a lot of mobility this would be an important factor. Therefore, a series of outdoor tests was performed where messages were exchanged between two devices at varying distances. The distance was increased from 10 m to 100 m in 10 m increments. As Fig. 5.5 clearly shows, the distance between the devices can be disregarded under line-of-sight (LOS) conditions.

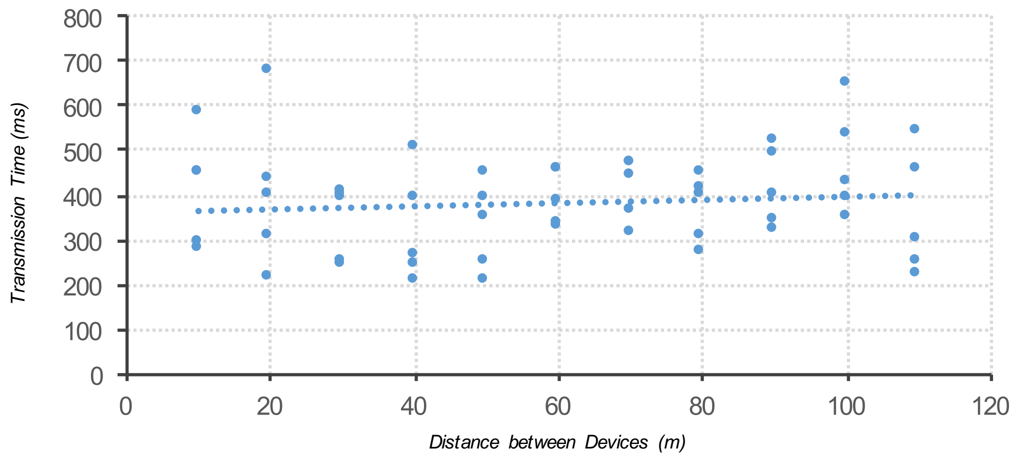


FIGURE 5.5: Message transmission times for different distances between sender and receiver

However, the situation looks completely different when there is no LOS between the two devices. Especially, in indoor scenarios, a smartphone's ability to act as an access point over longer distances can be severely weakened due to dense multi-path effects in the environment and reflection, diffraction and scattering of the WiFi signals [97]. While the evaluation scenario is focused purely on outdoor situations, one still shouldn't make the mistake to assume that smartphones will be able to communicate via opportunistic networking over distances of 100 m or more in the average case as it's very likely that there are no LOS conditions (please also see the next subsection for details on this topic).

(E) The Effect of dense Crowds

In section 5.2 it was elaborated that WiFi signals are severely attenuated by the human body. This fact is very relevant in the scope of the evaluation domain where large crowds are a given certainty and must therefore lead to meaningful reductions in the distance that can be covered by WiFi-based Ad-Hoc connections between smartphones. Therefore, it must not be assumed that during large scale events the same distances could be covered by the micro networks as they were observed during the LOS experiments. Tests in crowded open air environments have shown that the smartphones carried inside of trouser pockets were still able to communicate over a distance of approximately 50m but would sporadically lose their connection when the test subjects moved away even further. As a consequence, the simulation used for refining the parameters for the mobility-based mode switching strategy is assuming a maximum communication distance of 50 m between devices.

(F) Exchange Strategy

The communication within a micro-network consisting of one device in access point mode and several devices in client mode can be described best as a synchronization between all participating devices which is controlled by the access point device. The question is consequently, if synchronization should happen via flooding (i.e. all clients send all their messages to the access point which in turn sends the combined set back to all clients) or if it makes sense for the synchronization to take place in a filtered way. The latter would introduce an additional stage taking place before the actual synchronization where each client sends the access point a message containing the IDs of all the messages the client has stored. The access point would then return a list containing only those message IDs that are unknown to the access point so that the amount of transferred data can be optimized. Under ideal conditions (i.e. no duplicate messages exist in the micro-network), experiments have shown flooding to be about 15 % faster than the filtering approach as all messages need to be transferred, anyway, and the filtering step is just unnecessary overhead. However, since such an ideal scenario is not realistic under real-world conditions, it was nevertheless decided to implement the filtering mechanism in the communication protocol in order to be able to cope with situations where there might be many large duplicate messages existing on the connected devices.

(G) Access Point Performance

During the experiments it became apparent that smartphone vendors actually limit their devices when it comes to the number of clients that can connect to an access point device simultaneously. This limit does not seem to depend on hardware factors alone as it was discovered that identical phones running different operating systems (stock Android vs. the Cyanogen Mod for example) imposed different limitations on the access point feature. After having analyzed 14 different phones in total, the conclusion was made that it is safe to assume that each device can handle 5 clients when in access point mode – an important number when designing the mode switching strategy as it is another boundary condition for the simulation. However, it needs to be said that some of the test devices were able to handle as much as 13 connected clients, so in practice there will be micro-networks with more than just six participating devices.

(H) Mode Switching Duration

Critical performance parameters are the time it takes an access point device to provide clients with a WiFi hotspot and the time it takes a client to connect to an access point. Since the evaluation scenario has a large degree of user mobility, devices potentially don't have much time to connect to each other and synchronize before they are out of range again. Experiments showed that it can take a device anything between 5 and 10s to create a usable WiFi hotspot. On the other hand, client devices took a maximum of 5s to connect to an access point. Based on the dataset recorded at the 2013 Zurich Festival it was established that the average speed of a pedestrian at an event is rarely above 1 m/s under normal conditions. Assuming an average access point range of 50m and a total worst-case micro network setup time of 15s, this leaves enough time for information synchronization to occur even when the two devices are moving in opposite directions.

5.4 The Mobility-Based Mode Switching Strategy

After having established the technical limitations of smartphones with respect to opportunistic networking, the actual mode switching strategy can now be defined based on both those findings and the characteristics of the crowd management domain. This is being done in two parts: firstly, the basic mode switching strategy is introduced. Afterwards, its parameters are optimized in the scope of a large scale simulation.

5.4.1 Strategy Overview

In essence, the scenario assumes that a section of the communication network collapses under heavy load as it can happen during large scale events. Therefore, some parts of the visitors are in a “blackout area” unable to connect to the internet and thus unable to receive messages via the crowd sensing app. The idea is now, that the app realizes that there’s no connectivity which in turn leads to it enabling the Ad-Hoc communication feature trying to exchange messages in a peer-to-peer manner. Messages sent by the event organizers or emergency forces are received by people outside the “blackout area” and are carried into the area as those people move around. Once inside the affected area, the apps of those people can inject the messages into the Ad-Hoc network where they are propagated.

At this point, it is worth mentioning that this work concentrates deliberately on only handling messages. In theory, one could argue that the Ad-Hoc network could also be used to transport crowd sensing data out of the blackout area so that for example the live crowd density heat map visualization would also remain functional during partial network failures. However, in practice this is not feasible for two reasons:

1. **Latency**: within mobile Ad-Hoc networks there is no guaranteed delivery time. Potentially, it can take several minutes until data is synchronized through the entire blackout area – which is the worst-case requirement for maximizing the chances of one user walking outside of that area and therefore carrying the data into a region with network connectivity where the data can finally be uploaded. It therefore stands to reason that the chances are very low that crowd sensing data would make it to the servers in time for being available for a live visualization.

2. **Volume of data**: in a worst-case scenario (from a data volume point of view), every app user within the blackout area is also contributing data to the crowd sensing system. Consequently, the Ad-Hoc network would be tasked with synchronizing upload data of each and every user once every minute. Clearly, this is very likely to

cause an overload on the smartphone-based network. In addition to that, it would mean that users carrying the data out of the blackout area would be required to upload all that data to the servers – which in turn would put considerable strain on their devices' batteries and their data plans.

As a consequence of these two considerations, the idea of also realizing crowd sensing features via Ad-Hoc networking was quickly dropped. Having said that, the Data Collection Platform's app component does of course ensure that all recorded data is uploaded to the Data Processing Back End once the devices are online again. Hence, the post-event functionality of the crowd sensing aspect is not impacted by network outages in any way.

As mentioned earlier, the proposed approach for delivering messages to those parts of a crowd currently affected by a network outage takes the works of [95] as a starting point. The underlying idea is that some smartphones enable their built-in WiFi hotspot mode, while others connect to those access points as clients. Information can then be exchanged within those micro networks. If the modes of access point and client are frequently changed, it leads to an eventual propagation of the information to all participating devices – data is consequently being transmitted in a multi-hop fashion (again, please see Fig. 5.1 for an illustration of that process).

At this point it is worth mentioning, that the decision was made deliberately against implementing other means of wireless communication. The WiFi Direct standard for example seems to be perfect for the given scenario as it enables device-to-device communication via WiFi. However, the vast majority of the devices available today does not support this standard [93]. Bluetooth on the other hand was ruled out for two reasons: firstly, the necessary pairing process can be quite cumbersome which is a big obstacle in the large-scale event scenario where the aim is to have little or no user interaction at all. Secondly, smartphone apps which want to communicate via Bluetooth are required by the operating system to ask their users for access to the Bluetooth hardware – it stands to reason that many users may be irritated by such a poll. Given that the envisioned approach is supposed to be supported by as many devices as possible, it was therefore decided to use standard WiFi as it is available on most devices currently available on the market.

The crucial element defining the efficiency of an opportunistic networking implementation is the mode switching strategy. A good strategy should ensure that the devices communicate in a way that maximizes the amount of new messages being exchanged within each micro-network. Hence, the case where the same devices are part of the same micro network in consecutive mode switching iterations should be avoided if possible.

Taking these goals as a basis, the proposed mobility-based mode switching strategy is based on the following two assumptions:

1. Devices with a high mobility carry potentially more new messages than those with a low mobility. The reasoning is intuitive: if a device traveled a long distance, it potentially carries messages from a source that's physically further away from it's current location. Since opportunistic networking is mainly about bridging physical distances as quickly as possible, device mobility should clearly be regarded as an important factor. Therefore, the quality attributes *device mobility* and *device mobility threshold* are introduced as DM and DM_{thresh}, respectively.

2. Ideally, each micro network consists of devices that haven't been connected for a long time. Again, the reasoning is rather intuitive: the likelihood of new and therefore relevant messages being exchanged between devices in an opportunistic network is higher when the devices haven't been in touch with one another for a long time, since they will have had the chance to collect new messages in the meantime. As a consequence of this, the quality attributes *last seen* and *last seen threshold* are introduced as LS and LS_{thresh}, respectively.

The proposed mode switching strategy follows a strictly asynchronous approach: as soon as a device loses connectivity for a certain amount of time, it enters into the opportunistic networking mode and initiates a so-called **Decision Phase** where it decides whether it becomes an access point or a client without complying with a global schedule like it is done in [103]. Within that decision phase, the following main steps are taken (please also see Fig. 5.6 for reference):

1. The device collects information about all available access points that are within reach. Access point devices broadcast their DM value and their device name as part of their SSID.

2. All access points with $DM \geq DM_{thresh}$ are inserted into a list L_{DM} which is ranked according to the respective DM values.

3. The access points' LS values are computed based on their device names as each device keeps a local record of the time when it was last connected to another device. Afterwards, all access points with $LS \geq LS_{thresh}$ are inserted into a list L_{LS} which is ranked according to the respective LS values.

4. An overall ranking is computed based on each access point's ranking within L_{DM} and L_{LS}. Both of those rankings can be weighted – the optimal weight factors DM_{weight} and LS_{weight} are discussed in section 5.4.2.

5. If a suitable access point has been found, the device goes into client mode and connects to the access point with the highest overall ranking.

6. If no suitable access point could be found (because either none was available or none exceeded DM_{thresh} or LS_{thresh}), the device collects information about all available devices that are within reach.

7. If there are at least n devices within reach, the device goes into access point mode and waits for clients to connect. Please note that due to the limitations of some devices discovered during the initial experiments (see section 5.3 for details), n should not be greater than 5. An optimal value for n is discussed in section 5.4.2.

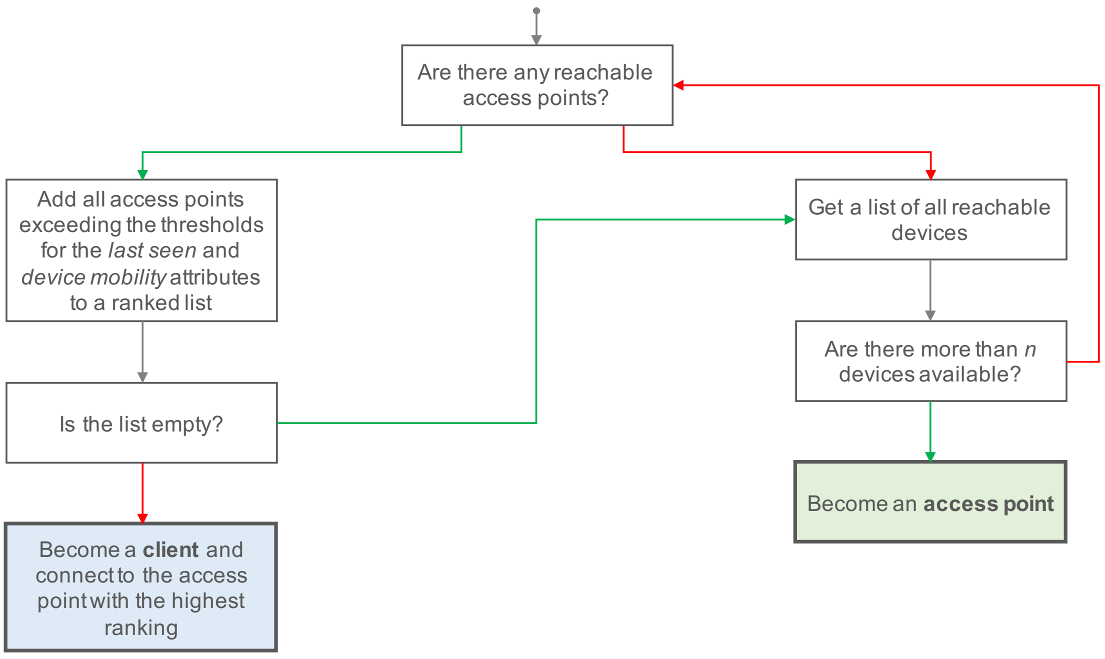

Are there any reachable access points?
Add all access points exceeding the thresholds for the *last seen* and *device mobility* attributes to a ranked list
Get a list of all reachable devices
Are there more than *n* devices available?
Is the list empty?
Become an **access point**
Become a **client** and connect to the access point with the highest ranking

FIGURE 5.6: Illustration of the mode switching strategy's decision phase *(green arrows indicate "yes", red arrows indicate "no")*

The devices also need to be able to make a decision about their modes, when the parameters DM_{thresh}, LS_{thresh} and n are not met by their surrounding peers. In those cases the decision phase is repeated, this time with lower values for DM_{thresh}, LS_{thresh} and n, effectively relaxing the requirements regarding the quality attributes a little bit. Furthermore, with each iteration of the decision phase, a random factor is added to the decision process so that after a large enough number of iterations, the decision basically defaults to a random mode switching strategy. In those extreme cases, 20% of the devices become access points while the rest goes into client mode. The reasoning for this 20/80 distribution is again found in the limitations of some smartphones with respect to their access point performance. This upper bound was chosen instead of a distribution with a higher access point ratio for battery drainage reason as devices in access point mode cause more battery drain than those in client mode [104].

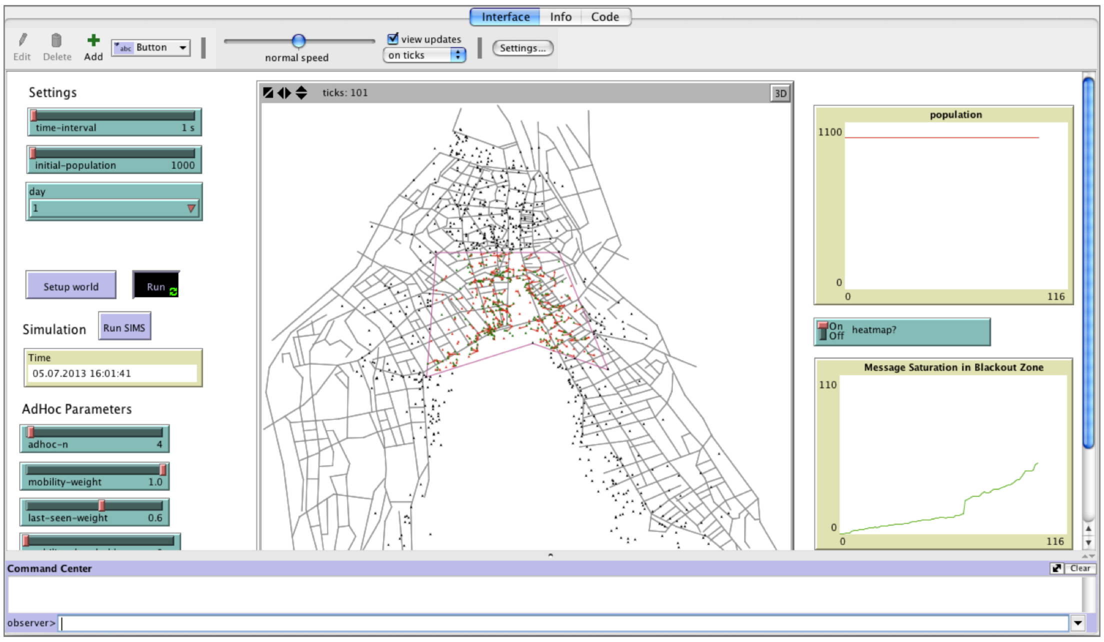
An integral element of the mode switching strategy is the definition of the point in time when a device currently acting as an access point should quit its current mode and go back into the decision phase. The proposed aproach makes access points responsible for breaking up connections – clients stay connected to an access point until that access point decides to go back into a decision phase. This happens when a certain amount of time has passed without any data being exchanged within the micro network – it was decided against a pre-defined global duration for the existence of access points as it can always be the case that new clients connect to an access point and deliver potentially new data to the micro network. Only when no data exchange has taken place for a set duration U is the micro network dissolved by the access point going back into a decision phase which triggers the clients to do the same. For further elaboration on the value U please see section 5.4.2. Naturally, clients are automatically disconnected from a micro network when they move out of their current access point's range. Should this be the case, the client enters a new decision phase automatically.

5.4.2 Parameter Optimization

After having formulated the proposed mobility-based mode switching strategy, its parameters need to be optimized. This has been done with the help of a large scale simulation based on the movement model created in the previous chapter. For the purpose of this task, the simulation environment has been extended in a way that allows users to define a blackout area on the map – this area represents the region that is currently affected by the connectivity outage. Furthermore, the concept of Ad-Hoc message synchronization utilizing the mobility-based mode switching strategy has been added to the implementation of the virtual agents.

The scenario to be examined is the propagation of a single message within the blackout area. In essence, it is therefore a variant of an infectious disease simulation – only that in this case, the "disease" is the spreading of a message. When a simulation run is started, all agents are placed on the map of Zurich randomly with their general distribution being based on the macroscopic movement model. Agents outside the blackout area are considered as "informed", i.e. they have received the message. Agents inside the blackout area are "uninformed" as they didn't receive the message, yet. When the simulation is started, the agents start walking over the map as defined in the microscopic movement model.

Now, if an informed agent walks into the blackout area, it starts the Ad-Hoc routine which leads to the message being propagated amongst the uninformed agents. In order to assess the quality of a given parameter configuration, the *saturation time* is measured, i.e. the time it takes until all uninformed agents have received the message. In the context of this work, "full saturation" has been defined as a state where at least 98 % of the uninformed agents got the message. The reason for this is that a full 100 % is never really reached, because the simulation constantly adds new agents within the blackout zone simulating new arrivals from a train station. Figure 5.7 shows a screenshot of the simulation environment. Here, the blackout area is defined by the pink polygon. Red agents inside this region represent agents whose devices are currently acting as access points, while green agents represent client devices.



time-interval 1 s
initial-population 1000
day 1
Time 05.07.2013 16:01:41
adhoc-n 4
mobility-weight 1.0
last-seen-weight 0.6
heatmap? On Off

FIGURE 5.7: Screenshot of the simulation engine used for the parameter optimization

For the actual parameter optimization process, a total of 15'552 simulation runs have been performed whereas each run was initialized with a population of 1'000 agents and a certain parameter configuration. The running time of the simulations was limited to 8 simulated minutes, thereby making sure that bad parameter configurations wouldn't cause extreme (or theoretically infinite) execution times for those particular runs. During each run, the message saturation was recorded every 10 simulated seconds.

Over the course of all simulations, a complete parameter sweep was performed for the variable parameters mentioned in the previous section. In summary, those are the uptime of micro networks (U), the minimum number of devices that need to surround another device for the latter to go into access point mode (n), the device mobility threshold (DM_{thresh}), the weight of the device mobility for the overall ranking ($weight_{DM}$), the last seen threshold (LS_{thresh}) and the weight of that last seen value for the overall ranking ($weight_{LS}$). Table 5.1 provides an overview about the range of each of those parameters and the corresponding step sizes. In addition, the best and the worst configurations are given.

Parameter	**Range Sweep Start**	**Range Sweep End**	**Range Sweep Step Size**	**Best Config.**	**Worst Config.**
U	60 s	180 s	30 s	60 s	180 s
n	1	4	1	4	2
$weight_{DM}$	0	1	0.1	1.0	0.2
DM_{thresh}	50 m	500 m	50 m	0 m	500 m
$weight_{LS}$	0	1	0.1	0.6	0.2
LS_{thresh}	0 s	300 s	0.1 s	0 s	60 s

Table 5.1: Ad-Hoc parameter optimization overview

Looking at those numbers in more detail reveals some interesting insights. For example, it can be seen that the proposed methodology performs best when the micro networks are dissolved sooner rather than later, thereby negating the initial assumption that in mobile scenarios it would be very likely for new devices to connect to the micro network after the "original participants" have exchanged their information. A very surprising finding was the fact that the device mobility value should be weighted almost twice as much as the last seen value when creating the ranking of access points during the decision phase.

With hindsight, it seems intuitive that the best performance is to be achieved when the threshold parameters of both the device mobility and last seen values are set to zero. In essence, this means that in the absence of an optimal access point, it is more efficient to simply connect to the best available one right away rather than going through several iterations within which the criteria for being an optimal access point are reduced. Finally, it is also clear that the mode switching strategy performs best, when access points are only created in the presence of as many other devices as possible.

Figure 5.8 shows the outcome of two simulations runs. The green curve represents the best parameter configuration while the red curve represents the worst configuration. It can be seen that in case of the former, full saturation was reached after only 150 seconds, while in case of the latter the saturation reached only a level of 60 % when the simulation was stopped after 8 simulated minutes.

Clearly, the entire concept of message distribution by means of Ad-Hoc networking is only feasible when there are enough participants for micro networks to form in the first place. Here, the crucial question is of course how the number of participants influences

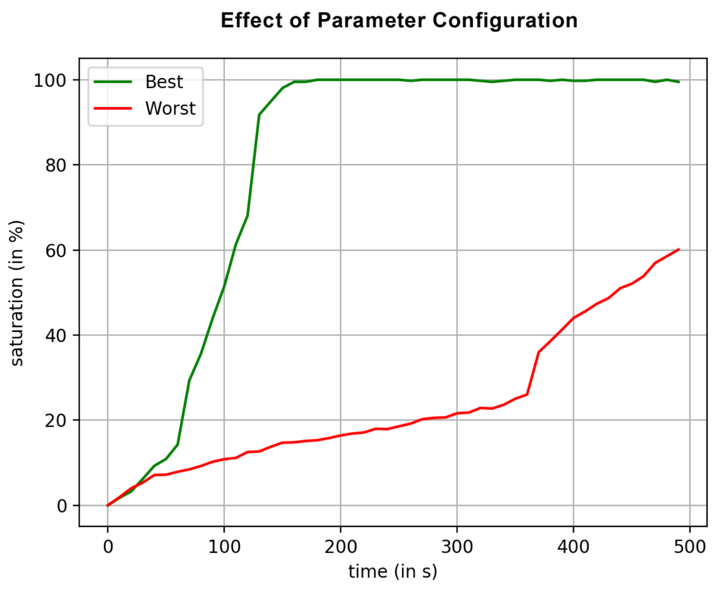


FIGURE 5.8: Comparison of the best and the worst parameter configuration

the saturation time within urban scenarios. Therefore, a number of simulation runs using the optimal parameter configuration has been performed for population sizes of 1'000, 5'000, 10'000, 28'000 and 40'000 participants. In Fig 5.9 it can be seen that while a scenario with a population of 1'000 does indeed lead to slightly slower saturation times, the differences aren't enormous. In fact, it seems to be safe to state that there aren't any noticeable performance gains once a population of 5'000 has been achieved. Therefore, the conclusion can be made that the proposed Ad-Hoc networking approach could indeed be successfully deployed during a scenario like the Zurich Festival.

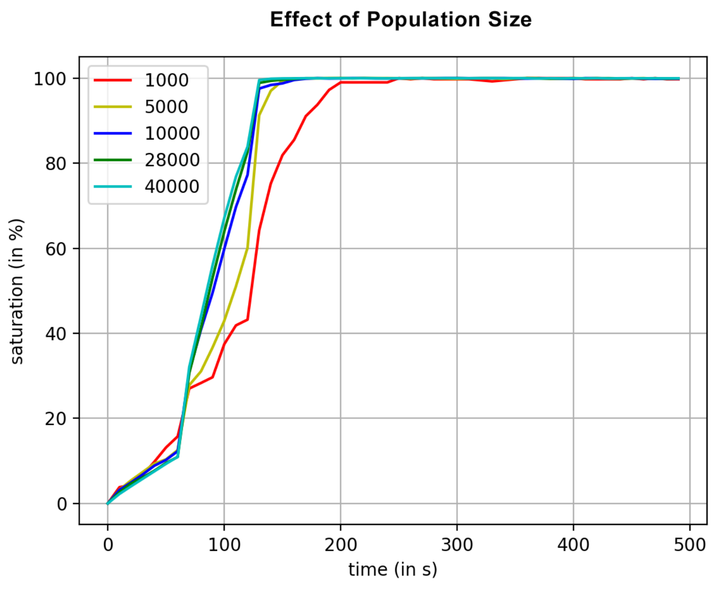


FIGURE 5.9: Effect of population size on saturation time

5.5 Evaluation

As a last step in this chapter, the performance of the proposed mobility-based mode switching strategy is compared to the performance of a traditional random mode switching strategy. For this, a total of 200 simulation runs have been performed – 100 runs using the mobility-based strategy with the best parameter configuration and 100 runs using a traditional random approach. In case of the latter, on average 20 % of the devices become access points after a micro network is broken up. Also, the clients randomly connect to any available access point without considering any quality attributes. Figure 5.10 compares the saturation times of the two approaches.

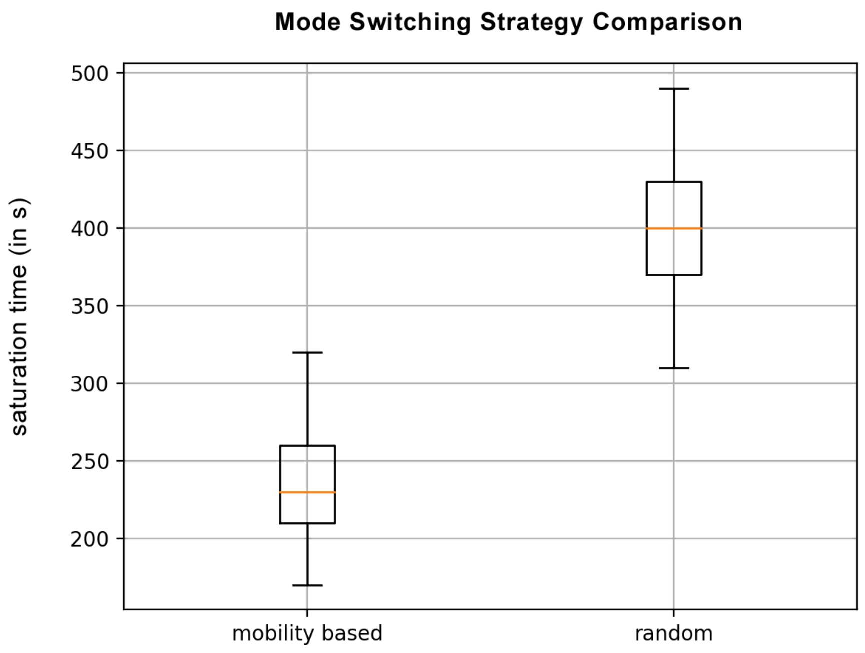


FIGURE 5.10: Comparison of the saturation times achieved when employing a mobility-based and a random mode switching strategy

Looking at all simulation runs, the saturation time in case of the mobility-based approach was between 90 s in the best case and 320 s in the worst case with an average of 230 s. The random approach meanwhile produced a best case saturation time of 310 s and a worst case of 480 s with an average of 400 s. Hence, it can be concluded that the mobility-based approach is on average more than 40 % quicker than the random approach.

The rather large spread of the saturation time values for both approaches can be explained by the fact that each simulation run generates slightly different crowd movements. Consequently, some scenarios might be more challenging for opportunistic networking than others.

5.6 Chapter Summary

This chapter provided a practical use case for the entire “crowd sensing framework” that has been developed over the course of this thesis. A pedestrian movement model – which was created based on the methodology presented in chapter 4 using data recorded with the Data Collection Platform introduced in chapter 2 and evaluated in chapter 3 – was used to refine the mode switching strategy of an Ad-Hoc communication system for usage during times of partial network outages at large scale events.

Before the mode switching strategy could be introduced, the boundary conditions imposed by smartphones with respect to Ad-Hoc networking were discussed. This was done by means of an empirical analysis which investigated and benchmarked the networking capabilities of a whole host of smartphones. The results were used to define the communication protocol and some fixed parameters of the proposed mode switching strategy (e.g. the maximum number of devices participating in a micro Ad-Hoc network).

Afterwards, the actual mode switching strategy was introduced. This was done by elaborating on the consequences arising from the characteristics of the highly mobile deployment scenario. Afterwards, the core algorithm of the strategy was presented. The exact behavior of this strategy is influenced by a number of parameters which would be very hard to set correctly using deterministic strategies. Therefore, the parameters were optimized in the course of a large scale simulation consisting of more than 15’000 simulation runs within which the best parameter set was determined. This simulation was a direct result from the works previously presented in the course of this thesis as it was based on the movement model presented in the previous chapter which in turn was created using data recorded with the Data Collection Platform during one of its evaluation scenarios.

The final contribution of this chapter was an evaluation of the proposed mobility-based mode switching strategy. It was shown that the mobility-based approach led to message distribution times which were on average 40 % faster than the ones that were achieved using traditional random approaches.

In summary, this chapter successfully demonstrated that the concept of crowd sensing cannot only be used for analysis tasks but also provides value for the development of complex algorithms and systems. While normally, such systems would be developed using deterministic approaches or generalized simulations, the crowd sensing approach allows for using models which are describing the special characteristics of a scenario very well and can therefore be used to tailor the system to those specifics. This in turn leads to a better performance compared to general-purpose solutions.

Chapter 6

Conclusion

"I may not have gone where I intended to go, but I think I have ended up where I needed to be."

Douglas Adams,
The Long Dark Tea-Time of the Soul

In this final chapter, the results and contributions of this thesis are summarized. Furthermore, the presented work is critically assessed with respect to both its relevance and its limitations. Finally, an outlook regarding potential technical and methodological future developments is given in order to provide a perspective on the next steps that need to be taken to advance the presented work.

6.1 Contributions Overview

In the introductory chapter, the following four research questions were formulated in order to provide an overview about the work covered in the course of this thesis:

1. What sort of architecture allows for quick and efficient deployments, scalability and adaptability of crowd sensing applications in a broad range of scenarios while at the same time enabling the integration of incentivation mechanisms for the participating public?
2. Are deployments of the proposed architecture feasible in the real world?
3. How can data models be extracted from the recorded data and are they representing the inherent core characteristics of the source data correctly?
4. How can complex smart services be developed based on those abstract data models?

With respect to question (1), chapter 2 presented the architecture of the proposed Data Collection Platform consisting of a Generic App Framework, a Data Receiving Back End and a Content Creation Back End in detail for each component. It was also shown how the overall system was designed to be scalable and easily extendable in order to be used for a wide range of crowd sensing applications in the future. In comparison to existing crowd sensing frameworks, it was shown that the proposed platform enables quick deployments – even for large scale scenarios – and has the unique capability to also provides end users with a concrete value proposition which is a key requirement for obtaining a large user base. In summary, the main contribution of the thesis' second chapter was the creation of an integrated Data Collection Platform specifically designed to cover the most important aspects of real-life, large-scale deployments in the Smart City domain: efficient setup, scalability, extendability and end-user value.

In order to demonstrate that real-world deployments of the proposed Data Collection Platform are in fact feasible – and therefore to answer research question (2) from above – chapter 3 presented an evaluation of the proposed Data Collection Platform within the domain of crowd management at large scale events. It was shown how an integrated crowd management system was built based on the Data Collection Platform which could be used to both analyze crowd characteristics like density, movement speed and movement direction in real time and to send context specific, geographically targeted messages to event visitors in order to exert crowd control. The resulting system was then shown to be feasible both in theoretical and in practical terms. The latter was achieved by performing a user study over the course of numerous system deployments with more than 100'000 end users interacting with the system. The outcomes of the study showed that both the end users and the event managers generally considered the system as a valuable tool that's well suited for its purpose. While the main contribution of the thesis' third chapter was in summary to proof that the Data Collection Platform is indeed suitable for being used productively in the real world, it also provided insights into soft factors that affected the outcomes of the deployment, therefore providing the research community with valuable insights "from the trenches".

After having demonstrated that the proposed Data Collection Platform has proven itself to be usable in the real world, chapter 4 explored the means to extract the inherent knowledge that's contained within crowd sensing data. For the purpose of answering the above research question (3), a methodology for extracting abstract models from data recorded at platform deployments in the crowd management domain was presented. Here, the main contributions were a spatio-temporal abstraction for dividing a city into appropriate context-related cells and a derivation of abstract crowd motion characteristics within each cell from a crowd sensing data set. The resulting model was then used to create a simulation of crowd movements during a large-scale city-wide festival. The final contribution of this fourth thesis chapter was to show that the methodology can be used to create a data model based on training data from two event days to then simulate the proceedings of a third day which may have a different geographical setup or event schedule. In the real world, this means that data recorded at a certain scenario can be used for more than just analyzing that particular scenario – it can also be used to create simulations which allow for observing how changes in a scenario's configuration can affect its outcome.

The main contribution of chapter 5 was to demonstrate how the concepts and technologies presented in this thesis could be used to create new applications (thereby answering research question (4) from above). This was again done by using the domain of crowd management as a show case. It was demonstrated how the crowd movement model created in chapter 4 drove a simulation which was used to optimize the parameters of a novel algorithm. The algorithm that was developed for this example was responsible for controlling the behavior of devices in an Ad-Hoc network responsible for enabling event visitors to receive messages from event organizers during times of partial network outages. The resulting algorithm was tailor made for this specific use case. This was achieved both by using the data-driven development approach and by performing an empirical analysis examining the technical boundary conditions imposed by the capabilities of smartphones with respect to Ad-Hoc networking. By comparing the novel algorithm to a more traditional approach in the course of a large-scale simulation, it was shown that it performed more than 40 % better, thereby proofing that the methodology of creating specialized applications by utilizing crowd sensing data during the development process is in fact feasible.

In summary, the main contributions of this thesis were the creation of a crowd sensing platform which was shown to be perfectly feasible for handling large-scale deployments in the real world and the formulation of a methodology for extracting abstract data models from this crowd sensed data. Those models have furthermore been shown to be suitable for both simulating other configurations of the crowd sensing scenario and for developing new, specialized applications for that scenario. Consequently, this work attempts to move the concept of crowd sensing from the research lab into the real world by demonstrating the width of its applications within the Smart City field in general and the crowd management domain in particular.

6.2 Relevance and Limitations

As mentioned in the introductory chapter, the global Smart City market is estimated to reach a level of 1.5 trillion US$ within the next years and therefore represents a very promising growth sector for the field of computer science. The majority of services in that sector – both present and future – are (or will be) based on large amounts of data. While some of that data is already existing in form of data bases (e.g. census data, etc.), a sizable part of it needs to be recorded by sensors. Depending on the exact application, it is of course possible to install dedicated sensor networks. However, for use cases where creating such fixed sensor networks is either too expensive or too time consuming, crowd sensing can certainly be a feasible alternative. Examples for such applications have been shown in section 1.2. Also, the work presented in this thesis certainly emphasizes the potential of crowd sensing as all the feedback gathered during the evaluation of the numerous deployments was positive.

A further point underlining the relevance of the concept of crowd sensing – particularly in the domain of crowd management – is the fact that some of the corporate research partners (e.g. Wembley Stadium) have explicitly suggested founding a company for commercializing the approach. Therefore, the startup SIS Software GmbH was founded in 2012 whose products are built on the foundation laid by the work presented in this thesis. So far, that company's clients are both from the event management domain (e.g. the Lord Mayor's Show) as well as from the public safety sector (e.g. the Dutch National Police) suggesting a wider market than “just” large scale events.

With respect to limitations, the first point that needs to be mentioned is of course that crowd sensing completely depends on the participation of end users – without enough participants, the recorded data is not representative and therefore more or less useless. The two key requirements for ensuring a large enough user base have been shown to be user incentivation and solid PR work. With respect to the former, it is absolutely vital that the end-users who are expected to contribute data to the system are seeing a personal benefit in their participation. Chapters 2 and 3 of this work mentioned some feasible approaches (e.g. gamification or access to exclusive information). Section 3.5.3 also elaborated on the topic of good PR work, which naturally, is a topic that cannot be tackled by computer scientists but instead has to be addressed by the deployment owners.

Crowd sensing is of course a topic which involves a whole multitude of stake holders. As shown in section 3.5.3, some of those are not always obvious. In the example, a major deployment of national importance was prevented because of a large telecommunications provider claiming that the communication load would be too much for its network to handle. While that statement – which turned out to have been made largely for commercial reasons – was proven to be incorrect eventually, it didn't change the negative outcome of the crowd sensing deployment. Section 6.3 suggests some developments which may help with also bringing telecommunication providers "on board" in order to prevent similar scenarios in the future.

A final potential limitation of the concept of crowd sensing is its dependency on the manufacturers of smartphone operating systems. If those should one day decide to disallow crowd sensing apps as continuously running background tasks, the concept will stop working. Apple for example has the very strict guideline that the app user has to have some form of benefit from the fact that an app is continuously using the location data service. During some of the evaluation deployments, app reviewers did in fact get in touch to ask where exactly the user benefit of the requested background sensing was. While those questions could always be answered to Apple's satisfaction, the example still shows that the operating system vendor has the power to potentially prevent a crowd sensing deployment. Again, this shows that it's of utmost importance to include features focused on providing value to the end-user into crowd sensing apps.

With respect to the proposed concept of Ad-Hoc communication, there are naturally also some limitations. Firstly, the impact of opportunistic P2P networking – especially when a device is in server mode – is not inconsiderable as elaborated by [104]. Furthermore, a true cross-platform implementation of this approach is currently prevented by Apple restricting access to some vital system level APIs. In addition to that, the smartphones' ability to create a WiFi hotspot can be limited by the telecommunication provider. In some cases, users have to pay a monthly fee to enable the feature. However, regardless of those current shortcomings, it seems to be clear that Ad-Hoc networking will play a big role in the future (e.g. in the crisis communication domain), especially given that new standards like WiFi Direct are about to be rolled out to mass production devices.

6.3 Outlook

Naturally, the work presented in this thesis can be further developed and optimized. Therefore, this final section provides an overview about potential next steps both in technical and methodological terms.

In section 3.5.4 it was shown that the app usage is in fact a precursor to developments in crowd size (see Fig. 3.11 for reference). As a logical next step, the app usage could be evaluated alongside the crowd sensed location data in real-time to provide deployment owners with insights on potential changes in crowd size. Taking this step even further, it would be feasible to also evaluate the logs regarding points of interest and navigation requests as those are directly targeting a specific location. If for example, a sudden peak in interest for a certain concert stage is observed, it stands to reason that within the near future a rise of the crowd size at that location is likely.

With respect to predicting crowd movements, it would be interesting to implement approaches for predictions based on live data only. In [106] for example, an online approach for crowd behavior prediction at a city-wide level is proposed. The authors manage to predict movements during the New Year's Eve celebrations in Tokio with a reasonable precision by clustering recent movement observations. Incorporating such approaches into the Data Collection Platform in the crowd management domain would certainly lead to a very relevant new feature for event organizers.

In the previous section it was mentioned that further developments of the platform are required in order to satisfy the needs of telecommunication providers which claim to be mainly concerned about network congestion. In order to address these concerns, it would be possible to implement a tool that allows providers to mark certain regions of the deployment area and define a limit with respect to the number of uploads per minute. While this might decrease the system's spatial resolution within those areas, this approach is still preferable to a situation where the deployment is being completely prevented by a telecommunication provider's veto.

With respect to the data modeling and simulation methodology, one of the biggest areas with room for improvement is the creation of a data structure that allows for more degrees of freedom with respect to movement of the agents than the street graph. This graph is too restrictive as it would only allow agents to walk along its edges representing streets or borders of open spaces. Consequently, the current methodology doesn't restrict the agents' movements at all and lets them walk around freely, which sometimes leads to behavior artifacts which would normally not be possible due to the city's infrastructure layout. For example, it was sometimes observed that agents were taking shortcuts which didn't exist in the real world. If a data structure was created allowing for more degrees of freedom than the street graph but less than the current open space implementation, it would improve the quality of the resulting simulations even further.

Finally, in section 3.3.3 it was elaborated that the spatial resolution of the crowd density estimation is hardly improving after a certain percentage of crowd sensing participants has been reached. If a deployment should actually be faced with the situation that there are "more participants than necessary", it would be advisable to implement a sub-sampling algorithm. Essentially, such an algorithm would ensure that only a certain percentage of all users would actually upload data at any given time. On average, this would further decrease the battery drain caused by crowd sensing apps which in turn would be a further incentive for users to participate.

In summary, this thesis proofed that the concept of crowd sensing is definitely ready to make the step from research labs and field tests into the real world. It was shown how truly large-scale deployments were realized leading to a system substantially raising the safety level of festivals while at the same time increasing the quality of the visitors' experience at the events. Given the fact that those deployments were realized on a research group's comparatively small budget, the next step should be a coordinated approach involving multiple agencies including city administrations which could lead to long-term deployments creating new insights for both improving the quality of urban life and for studying complex societal phenomena.

Bibliography

[1] Department of Economic United Nations and Population Division Social Affairs. World urbanization prospects: The 2014 revision, highlights. *World Urbanization Prospects*, 2014.

[2] Paolo Neirotti, Alberto De Marco, Anna Corinna Cagliano, Giulio Mangano, and Francesco Scorrano. Current trends in smart city initiatives: Some stylised facts. *Cities*, 38:25–36, 2014.

[3] Frost and Sullivan. Global smart cities market to reach us$1.56 trillion by 2020. `http://ww2.frost.com/news/press-releases/frost-sullivan-global-smart-cities-market-reach-us156-trillion-2020/`, 2014. [Online; accessed September 17, 2016].

[4] Milind Naphade, Guruduth Banavar, Colin Harrison, Jurij Paraszczak, and Robert Morris. Smarter cities and their innovation challenges. *Computer*, 44(6):32–39, 2011.

[5] Nicholas D Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T Campbell. A survey of mobile phone sensing. *IEEE Communications magazine*, 48(9):140–150, 2010.

[6] Andrew T Campbell, Shane B Eisenman, Nicholas D Lane, Emiliano Miluzzo, Ronald A Peterson, Hong Lu, Xiao Zheng, Mirco Musolesi, Kristóf Fodor, and Gahng-Seop Ahn. The rise of people-centric sensing. *IEEE Internet Computing*, 12(4):12–21, 2008.

[7] Statista. Prognose zum anteil der smartphone-nutzer in deutschland von 2012 bis 2018. `http://de.statista.com/statistik/daten/studie/321935/umfrage/prognose-zum-anteil-der-smartphone-nutzer-in-deutschland/`, 2014. [Online; accessed September 17, 2016].

[8] Statista. Mobile phone internet user penetration in china from 2014 to 2021. `http://www.statista.com/statistics/309015/china-mobile-phone-internet-user-penetration/`, 2016. [Online; accessed September 17, 2016].

[9] GSMA Intelligence. Half a billion chinese citizens have subscribed to the mobile internet. `https://www.gsmaintelligence.com/research/2014/06/half-a-billion-chinese-citizens-have-subscribed-to-the-mobile-internet/432/`, 2014. [Online; accessed September 17, 2016].

[10] Min Mun, Sasank Reddy, Katie Shilton, Nathan Yau, Jeff Burke, Deborah Estrin, Mark Hansen, Eric Howard, Ruth West, and Péter Boda. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 55–68. ACM, 2009.

[11] Emiliano Miluzzo, Nicholas D Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B Eisenman, Xiao Zheng, and Andrew T Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 337–350. ACM, 2008.

[12] Haggai Roitman, Jonathan Mamou, Sameep Mehta, Aharon Satt, and LV Subramaniam. *harnessing the crowds for smart city sensing. In *Proceedings of the 1st international workshop on Multimodal crowd sensing*, pages 17–18. ACM, 2012.

[13] Avik Ghose, Provat Biswas, Chirabrata Bhaumik, Monika Sharma, Arpan Pal, and Abhinav Jha. *road condition monitoring and alert application: Using in-vehicle smartphone as internet-connected sensor. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 489–491. IEEE, 2012.

[14] Bei Pan, Yu Zheng, David Wilkie, and Cyrus Shahabi. Crowd sensing of traffic anomalies based on human mobility and social media. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 344–353. ACM, 2013.

[15] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson. Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 85–98. ACM, 2009.

[16] Matthias Stevens and Ellie D'Hondt. Crowdsourcing of pollution data using smartphones. In *Workshop on Ubiquitous Crowdsourcing*, 2010.

[17] Irene Garcia Martí, Luis E Rodríguez, Mauricia Benedito, Sergi Trilles, Arturo Beltrán, Laura Díaz, and Joaquín Huerta. Mobile application for noise pollution

monitoring through gamification techniques. In *International Conference on Entertainment Computing*, pages 562–571. Springer, 2012.

[18] Rüdiger Pryss, Manfred Reichert, Jochen Herrmann, Berthold Langguth, and Winfried Schlee. Mobile crowd sensing in clinical and psychological trials–a case study. In *2015 IEEE 28th International Symposium on Computer-Based Medical Systems*, pages 23–24. IEEE, 2015.

[19] Apple. Apple introduces researchkit, giving medical researchers the tools to revolutionize medical studies. `http://www.apple.com/pr/library/2015/03/09Apple-Introduces-ResearchKit-Giving-Medical-Researchers-the-Tools-to-Revolutionize-Medical-Studies.html`, 2015. [Online; accessed September 16, 2016].

[20] Fosca Giannotti, Dino Pedreschi, Alex Pentland, Paul Lukowicz, Donald Kossmann, James Crowley, and Dirk Helbing. A planetary nervous system for social mining and collective awareness. *The European Physical Journal Special Topics*, 214(1):49–75, 2012.

[21] Tingxin Yan, Matt Marzilli, Ryan Holmes, Deepak Ganesan, and Mark Corner. mcrowd: a platform for mobile crowdsourcing. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 347–348. ACM, 2009.

[22] Bin Guo, Zhiwen Yu, Xingshe Zhou, and Daqing Zhang. From participatory sensing to mobile crowd sensing. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 593–598. IEEE, 2014.

[23] Prem Prakash Jayaraman, Charith Perera, Dimitrios Georgakopoulos, and Arkady Zaslavsky. Efficient opportunistic sensing using mobile collaborative platform mosden. In *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*, pages 77–86. IEEE, 2013.

[24] Moo-Ryong Ra, Bin Liu, Tom F La Porta, and Ramesh Govindan. Medusa: A programming framework for crowd-sensing applications. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 337–350. ACM, 2012.

[25] Giuseppe Cardone, Luca Foschini, Paolo Bellavista, Antonio Corradi, Cristian Borcea, Manoop Talasila, and Reza Curtmola. Fostering participaction in smart cities: a geo-social crowdsensing platform. *IEEE Communications Magazine*, 51 (6):112–119, 2013.

[26] Giuseppe Cardone, Andrea Cirri, Antonio Corradi, and Luca Foschini. The participact mobile crowd sensing living lab: The testbed for smart cities. *IEEE Communications Magazine*, 52(10):78–85, 2014.

[27] Murat Demirbas, Murat Ali Bayir, Cuneyt Gurcan Akcora, Yavuz Selim Yilmaz, and Hakan Ferhatosmanoglu. Crowd-sourced sensing and collaboration using twitter. In *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*, pages 1–9. IEEE, 2010.

[28] R Szabo, K Farkas, M Ispany, AA Benczúr, N Batfai, P Jeszenszky, S Laki, A Vagner, L Kollar, Cs Sidló, et al. *framework for smart city applications based on participatory sensing. In *Cognitive Infocommunications (CogInfoCom), 2013 IEEE 4th International Conference on*, pages 295–300. IEEE, 2013.

[29] Spyros Xanthopoulos and Stelios Xinogalos. A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics*, pages 213–220. ACM, 2013.

[30] Anton Dollmaier. Development of a highly available, redundant geodatabase system. Master's thesis, University of Passau, 2013.

[31] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2), 2010.

[32] Dirk Helbing, Lubos Buzna, Anders Johansson, and Torsten Werner. *self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. *Transportation science*, 39(1):1–24, 2005.

[33] Dirk Helbing, Illes J Farkas, Peter Molnar, and Tamás Vicsek. *simulation of pedestrian crowds in normal and evacuation situations. *Pedestrian and evacuation dynamics*, 21:21–58, 2002.

[34] Ka Ming Ngai, Frederick M Burkle, Anthony Hsu, and Edbert B Hsu. *human stampedes: a systematic review of historical and peer-reviewed sources. *Disaster medicine and public health preparedness*, 3(04):191–195, 2009.

[35] Dirk Helbing and Pratik Mukerji. Crowd disasters as systemic failures: analysis of the love parade disaster. *EPJ Data Science*, 1(1):1–40, 2012.

[36] Martin Wirz, Tobias Franke, Daniel Roggen, Eve Mitleton-Kelly, Paul Lukowicz, and Gerhard Tröster. Probing crowd density through smartphones in city-scale mass gatherings. *EPJ Data Science*, 2(1):1–24, 2013.

[37] Martin Wirz, Tobias Franke, Eve Mitleton-Kelly, Daniel Roggen, Paul Lukowicz, and Gerhard Tröster. Coenosense: A framework for real-time detection and visualization of collective behaviors in human crowds by tracking mobile devices. In *Proceedings of the European Conference on Complex Systems 2012*, pages 353–361. Springer, 2013.

[38] Hidayah Rahmalan, Mark S Nixon, and John N Carter. On crowd density estimation for surveillance. 2006.

[39] Anthony C Davies, Jia Hong Yin, and Sergio A Velastin. *crowd monitoring using image processing. *Electronics & Communication Engineering Journal*, 7(1):37–47, 1995.

[40] BPL Lo and SA Velastin. *automatic congestion detection system for underground platforms. In *Intelligent Multimedia, Video and Speech Processing, 2001. Proceedings of 2001 International Symposium on*, pages 158–161. IEEE, 2001.

[41] Ramin Mehran, Alexis Oyama, and Mubarak Shah. Abnormal crowd behavior detection using social force model. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 935–942. IEEE, 2009.

[42] Beibei Zhan, Dorothy N Monekosso, Paolo Remagnino, Sergio A Velastin, and Li-Qun Xu. Crowd analysis: a survey. *Machine Vision and Applications*, 19(5-6): 345–357, 2008.

[43] Shaogang Gong, Chen Change Loy, and Tao Xiang. Security and surveillance. In *Visual Analysis of Humans*, pages 455–472. Springer, 2011.

[44] Julio Silveira Jacques Junior, Soraia Musse, and Claudio Jung. Crowd analysis using computer vision techniques. *IEEE Signal Processing Magazine*, 5(27):66–77, 2010.

[45] Barbara Krausz and Christian Bauckhage. Loveparade 2010: Automatic video analysis of a crowd disaster. *Computer Vision and Image Understanding*, 116(3): 307–319, 2012.

[46] Antoni B Chan, Z-SJ Liang, and Nuno Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–7. IEEE, 2008.

[47] Bi Song, Ricky J Sethi, and Amit K Roy-Chowdhury. Wide area tracking in single and multiple views. In *Visual Analysis of Humans*, pages 91–107. Springer, 2011.

[48] Martin Wirz, Pablo Schläpfer, Mikkel Baun Kjærgaard, Daniel Roggen, Sebastian Feese, and Gerhard Tröster. Towards an online detection of pedestrian flocks in urban canyons by smoothed spatio-temporal clustering of gps trajectories. In *Proceedings of the 3rd ACM SIGSPATIAL international workshop on location-based social networks*, pages 17–24. ACM, 2011.

[49] Nabeel Koshak and Abdullah Fouda. Analyzing pedestrian movement in mataf using gps and gis to support space redesign. In *The 9th international conference on design and decision support systems in architecture and urban planning*, 2008.

[50] Robert Pettersson and Malin Zillinger. Time and space in event behaviour: Tracking visitors by gps. *Tourism Geographies*, 13(1):1–20, 2011.

[51] Jonathan Reades, Francesco Calabrese, Andres Sevtsuk, and Carlo Ratti. Cellular census: Explorations in urban data collection. *IEEE Pervasive Computing*, 6(3): 30–38, 2007.

[52] Francesco Calabrese, Massimo Colonna, Piero Lovisolo, Dario Parata, and Carlo Ratti. Real-time urban monitoring using cell phones: A case study in rome. *IEEE Transactions on Intelligent Transportation Systems*, 12(1):141–151, 2011.

[53] Richard A Becker, Ramon Caceres, Karrie Hanson, Ji Meng Loh, Simon Urbanek, Alexander Varshavsky, and Chris Volinsky. A tale of one city: Using cellular network data for urban planning. *IEEE Pervasive Computing*, 10(4):18–26, 2011.

[54] T Couronne, A-M Olteanu Raimond, and Z Smoreda. Looking at spatiotemporal city dynamics through mobile phone lenses. In *2011 International Conference on the Network of the Future*, 2011.

[55] Francesco Calabrese, Francisco C. Pereira, Giusy Di Lorenzo, Liang Liu, and Carlo Ratti. *The Geography of Taste: Analyzing Cell-Phone Mobility and Social Events*, pages 22–37. 2010.

[56] Tobias Franke, Paul Lukowicz, and Ulf Blanke. Smart crowds in smart cities: real life, city scale deployments of a smartphone based participatory crowd management platform. *Journal of Internet Services and Applications*, 6(1):27, 2015.

[57] Tobias Franke, Paul Lukowicz, Martin Wirz, and Eve Mitleton-Kelly. Participatory sensing and crowd management in public spaces. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 485–486. ACM, 2013.

[58] Ulf Blanke, Gerhard Troster, Tobias Franke, and Paul Lukowicz. Capturing crowd dynamics at large scale events using participatory gps-localization. In *Intelligent*

Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on, pages 1–7. IEEE, 2014.

[59] Martin Wirz, Tobias Franke, Daniel Roggen, Eve Mitleton-Kelly, Paul Lukowicz, and Gerhard Troster. Inferring crowd conditions from pedestrians' location traces for real-time crowd monitoring during city-scale mass gatherings. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*, pages 367–372. IEEE, 2012.

[60] Hendrik Stange, Thomas Liebig, Dirk Hecker, Gennady Andrienko, and Natalia Andrienko. Analytical workflow of monitoring human mobility in big event settings using bluetooth. In *Proceedings of the 3rd ACM SIGSPATIAL international workshop on indoor spatial awareness*, pages 51–58. ACM, 2011.

[61] Arkadiusz Stopczynski, Jakob Eg Larsen, Sune Lehmann, Lukasz Dynowski, and Marcos Fuentes. Participatory bluetooth sensing: A method for acquiring spatio-temporal data about participant mobility and interactions at large scale events. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*, pages 242–247. IEEE, 2013.

[62] Jens Weppner and Paul Lukowicz. Collaborative crowd density estimation with mobile phones. *Proc. of ACM PhoneSense*, 2011.

[63] Jens Weppner and Paul Lukowicz. Bluetooth based collaborative crowd density estimation with mobile phones. In *Pervasive computing and communications (PerCom), 2013 IEEE international conference on*, pages 193–200. IEEE, 2013.

[64] Jens Weppner, Paul Lukowicz, Ulf Blanke, and Gerhard Tröster. Participatory bluetooth scans serving as urban crowd probes. *IEEE Sensors Journal*, 14(12): 4196–4206, 2014.

[65] Mathias Versichele, Tijs Neutens, Matthias Delafontaine, and Nico Van de Weghe. The use of bluetooth for analysing spatiotemporal dynamics of human movement at mass events: A case study of the ghent festivities. *Applied Geography*, 32(2): 208–220, 2012.

[66] Anders Johansson, Dirk Helbing, Habib Z Al-Abideen, and Salim Al-Bosta. From crowd dynamics to crowd safety: a video-based analysis. *Advances in Complex Systems*, 11(04):497–527, 2008.

[67] Z Fang, SM Lo, and JA Lu. On the relationship between crowd density and movement velocity. *Fire Safety Journal*, 38(3):271–283, 2003.

[68] Dirk Helbing, Anders Johansson, and Habib Zein Al-Abideen. Dynamics of crowd disasters: An empirical study. *Physical review E*, 75(4):046109, 2007.

[69] David W Scott. *Multivariate density estimation: theory, practice, and visualization.* John Wiley & Sons, 2015.

[70] Chris Brunsdon and Martin Charlton. Local trend statistics for directional data—a moving window approach. *Computers, environment and urban systems*, 30(2):130–142, 2006.

[71] Dirk Oberhagemann. Statische und dynamische personendichten bei grossveranstaltungen. Technical report, Technical Report of the Association for the Improvement of German Fire Protection, 2012.

[72] Mehdi Moussaid, Simon Garnier, Guy Theraulaz, and Dirk Helbing. Collective information processing and pattern formation in swarms, flocks, and crowds. *Topics in Cognitive Science*, 1(3):469–497, 2009.

[73] Dirk Helbing, Illés Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[74] Stephen Wolfram et al. Cellular automata as models of complexity. *Nature*, 311 (5985):419–424, 1984.

[75] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[76] Hubert Ludwig Kluepfel. *A cellular automaton model for crowd movement and egress simulation.* PhD thesis, Universität Duisburg-Essen, Fakultät für Physik, 2003.

[77] Gerta Köster, Dirk Hartmann, and Wolfram Klein. Microscopic pedestrian simulations: From passenger exchange times to regional evacuation. In *Operations Research Proceedings 2010*, pages 571–576. Springer, 2011.

[78] Carsten Burstedde, Kai Klauck, Andreas Schadschneider, and Johannes Zittartz. Simulation of pedestrian dynamics using a two-dimensional cellular automaton. *Physica A: Statistical Mechanics and its Applications*, 295(3):507–525, 2001.

[79] Kashif Zia, Andreas Riener, Alois Ferscha, and Alexei Sharpanskykh. Evacuation simulation based on cognitive decision making model in a socio-technical system. In *Proceedings of the 2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications*, pages 98–107. IEEE Computer Society, 2011.

[80] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(suppl 3): 7280–7287, 2002.

[81] Yohei Murakami, Kazuhisa Minami, Tomoyuki Kawasoe, and Toru Ishida. Multi-agent simulation for crisis management. In *Knowledge Media Networking, 2002. Proceedings. IEEE Workshop on*, pages 135–139. IEEE, 2002.

[82] Anders Johansson, Michael Batty, Konrad Hayashi, Osama Al Bar, David Marcozzi, and Ziad A Memish. Crowd and environmental management during mass gatherings. *The Lancet infectious diseases*, 12(2):150–156, 2012.

[83] Kashif Zia, Alois Ferscha, Andreas Riener, Martin Wirz, Daniel Roggen, Kamil Kloch, and Paul Lukowicz. Scenario based modeling for very large scale simulations. In *Distributed Simulation and Real Time Applications (DS-RT), 2010 IEEE/ACM 14th International Symposium on*, pages 103–110. IEEE, 2010.

[84] Kashif Zia, Sajjad A Madani, Sardar Nauman Aslam, and Alois Ferscha. Self-organized togetherness in a crowd group. In *Proceedings of the 7th International Conference on Frontiers of Information Technology*, page 9. ACM, 2009.

[85] Andreas Poxrucker, Gernot Bahle, and Paul Lukowicz. Towards a real-world simulator for collaborative distributed learning in the scenario of urban mobility. In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2014 IEEE Eighth International Conference on*, pages 44–48. IEEE, 2014.

[86] Andreas Poxrucker, Gernot Bahle, and Paul Lukowicz. Simulating adaptive, personalized, multi-modal mobility in smart cities. In *Smart City 360*, pages 113–124. Springer, 2016.

[87] Michael Batty. *Cities and complexity: understanding cities with cellular automata, agent-based models, and fractals.* The MIT press, 2007.

[88] Bernhard Steffen and Armin Seyfried. Methods for measuring pedestrian density, flow, speed and direction with minimal scatter. *Physica A: Statistical mechanics and its applications*, 389(9):1902–1910, 2010.

[89] Ameya Shendarkar, Karthik Vasudevan, Seungho Lee, and Young-Jun Son. *crowd simulation for emergency response using bdi agent based on virtual reality. In *Proceedings of the 38th conference on Winter simulation*, pages 545–553. Winter Simulation Conference, 2006.

[90] Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[91] U. Wilensky. Netlogo. http://ccl.northwestern.edu/netlogo/, 1999. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[92] Mehran Abolhasan, Tadeusz Wysocki, and Eryk Dutkiewicz. A review of routing protocols for mobile ad hoc networks. *Ad hoc networks*, 2(1):1–22, 2004.

[93] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. Device-to-device communications with wi-fi direct: overview and experimentation. *Wireless Communications, IEEE*, 20(3), 2013.

[94] Franck Legendre, Theus Hossmann, Felix Sutton, and Bernhard Plattner. 30 years of wireless ad hoc networking research: What about humanitarian and disaster relief solutions? what are we still missing? In *Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief (ACWR), Amritapuri, Kollam, Kerala, India*, pages 217–217, 2011.

[95] Sacha Trifunovic, Bernhard Distl, Dominik Schatzmann, and Franck Legendre. Wifi-opp: ad-hoc-less opportunistic networking. In *Proceedings of the 6th ACM workshop on Challenged networks*, pages 37–42. ACM, 2011.

[96] Sacha Trifunovic, Maciej Kurant, Karin Anna Hummel, and Franck Legendre. Wlan-opp: Ad-hoc-less opportunistic networking on smartphones. *Ad Hoc Networks*, 2014.

[97] Kamol Kaemarungsi and Prashant Krishnamurthy. Properties of indoor received signal strength for wlan location fingerprinting. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pages 14–23. IEEE, 2004.

[98] Henry C Lukaski. Methods for the assessment of human body composition: traditional and new. *The American journal of clinical nutrition*, 46(4):537–556, 1987.

[99] Kamol Kaemarungsi. Distribution of wlan received signal strength indication for indoor location determination. In *Wireless Pervasive Computing, 2006 1st International Symposium on*, pages 6–pp. IEEE, 2006.

[100] Ngewi Fet, Marcus Handte, and Pedro José Marrón. A model for wlan signal attenuation of the human body. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 499–508. ACM, 2013.

[101] Daniel J Dubois, Yosuke Bando, Konosuke Watanabe, and Henry Holtzman. Lightweight self-organizing reconfiguration of opportunistic infrastructure-mode

wifi networks. In *Self-Adaptive and Self-Organizing Systems (SASO), 2013 IEEE 7th International Conference on*, pages 247–256. IEEE, 2013.

[102] Elmurod Talipov, Jianxiong Yin, Yohan Chon, and Hojung Cha. A context-rich and extensible framework for spontaneous smartphone networking. *Computer Communications*, 37:25–39, 2014.

[103] Mayank Raj, Krishna Kant, and Sajal K Das. E-darwin: Energy aware disaster recovery network using wifi tethering. In *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*, pages 1–8. IEEE, 2014.

[104] Sacha Trifunovic, Andreea Picu, Theus Hossmann, and Karin Anna Hummel. Adaptive role switching for fair and efficient battery usage in device-to-device communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 18(1):25–36, 2014.

[105] Tobias Franke, Sascha Negele, George Kampis, and Paul Lukowicz. Leveraging human mobility in smartphone based ad-hoc information distribution in crowd management scenarios. In *Information and Communication Technologies for Disaster Management (ICT-DM), 2015 2nd International Conference on*, pages 27–34. IEEE, 2015.

[106] Zipei Fan, Xuan Song, Ryosuke Shibasaki, and Ryutaro Adachi. Citymomentum: an online approach for crowd behavior prediction at a citywide level. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 559–569. ACM, 2015.

Curriculum Vitae

Tobias Franke received his diploma in computer science (area of specialization: intelligent embedded systems) from the University of Passau in 2009. Afterwards, he has been working as research assistant at the Embedded Systems Lab at the University of Passau. In 2012 he joined the German Research Center for Artificial Intelligence (DFKI) in Kaiserslautern. During his time in the research world, his work focused on the area of large scale cooperative sensing. He was involved in several European and international projects (e.g. SOCIONICAL, FuturICT and RESCUER) and also realized projects for large industry partners (e.g. Volkswagen and the City of London). In 2012 he founded the startup company SIS Software GmbH where he is still holding the CEO position.