UNIVERSITY OF CALIFORNIA,
IRVINE

Exploiting Mobile Plus In-Situ Deployments in Community IoT Systems

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Qiuxi Zhu

Dissertation Committee:
Professor Nalini Venkatasubramanian, Chair
Professor Nikil Dutt
Professor Marco Levorato
Professor Sharad Mehrotra

2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Nalini Venkatasubramanian, for all her guidance throughout the years.

I also thank my coauthors on the referred publications and various other peers that I worked closely with during my Ph.D.: Md Yusuf Sarwar Uddin, Zhijing Qin, Cheng-Hsin Hsu, Valerie Issarny, Francoise Sailhan, Chao-Wen Chen, Guoxi Wang, Kyle Benson, Qing Han, Fangqi Liu, and everyone in the Distributed Systems Middleware (DSM) group and the Information Systems Group (ISG) at the University of California, Irvine, for their significant contribution and/or valuable feedback.

I thank my various mentors from both my Ph.D. studies and internships: my dissertation committee – Professor Nikil Dutt, Professor Marco Levorato, and Professor Sharad Mehrotra from UCI; Daniel Hoffman from Montgomery County, MD; Dr. Greg Bollella and Dr. Hui Xu from VMware, Inc.

I further thank the various teams and collaborators who have either contributed to this research or supported it in some capacity: the SCALE team; the Networking and Multimedia Systems Lab (NMSL) at National Tsing Hua University (NTHU), Taiwan; the MINES associate team supported by the French Institute for Research in Computer Science and Automation (Inria); the National Fire Protection Association (NFPA).

My Ph.D. research is supported in part by the National Science Foundation (NSF) award No. CNS-1450768, the National Institute of Standards and Technology (NIST) award No. 70NANB17H285, and the Donald Bren School of Information and Computer Sciences (ICS) at the University of California, Irvine (UCI).

CURRICULUM VITAE

Qiuxi Zhu

EDUCATION

Doctor of Philosophy in Computer Science — **2019**
University of California, Irvine — *Irvine, California*

Bachelor of Engineering in Automation — **2013**
Zhejiang University — *Hangzhou, China*

RESEARCH EXPERIENCE

Graduate Student Researcher — **2014–2018**
University of California, Irvine — *Irvine, California*

Undergraduate Student Researcher — **2011–2013**
Zhejiang University — *Hangzhou, China*

PROFESSIONAL EXPERIENCE

Intern, Internet of Things — **Summer 2018**
VMware, Inc. — *Palo Alto, California*

Intern, Internet of Things — **Summer 2017**
VMware, Inc. — *Palo Alto, California*

Intern, Internet of Things — **Summer 2016**
VMware, Inc. — *Palo Alto, California*

Innovation Fellow — **Spring – Summer 2015**
Montgomery County Innovation Program — *Rockville, Maryland*

TEACHING EXPERIENCE

Teaching Assistant — **2016–2019**
University of California, Irvine — *Irvine, California*

Graduate Reader — **2014–2016**
University of California, Irvine — *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

Data collection and upload under dynamicity in smart community Internet-of-Things deployments **Dec 2017**
Pervasive and Mobile Computing

REFEREED CONFERENCE PUBLICATIONS

Multi-Sensor Calibration Planning in IoT-Enabled Smart Spaces **Jul 2019**
IEEE International Conference on Distributed Computing Systems (ICDCS)

Spatiotemporal Scheduling for Crowd Augmented Urban Sensing **Apr 2018**
IEEE Conference on Computer Communications (INFOCOM)

Upload Planning for Mobile Data Collection in Smart Community Internet-of-Things Deployments **May 2016**
IEEE International Conference on Smart Computing (SMARTCOMP)

SOFTWARE

SCALE client `https://github.com/KyleBenson/scale_client`
The Python-based SCALE client software for acquiring data from various sensors (i.e. via Raspberry Pi platform), processing it, and sharing it through multiple networks and data exchange protocols.

SCALECycle client `https://github.com/bfrggit/scale_client/tree/cycle`
A branch of the SCALE client with support for the SCALECycle features – GPS location, geo-fence, network connectivity awareness, local database, Bluetooth terminal event sink, task schedule, etc.

ABSTRACT OF THE DISSERTATION

Exploiting Mobile Plus In-Situ Deployments in Community IoT Systems

By

Qiuxi Zhu

Doctor of Philosophy in Computer Science

University of California, Irvine, 2019

Professor Nalini Venkatasubramanian, Chair

Improvements in Internet connectivity and advances in smart personal devices have enabled the rise of the Internet of Things (IoT) in real-world communities. Community IoT deployments utilize low-cost devices, often deployed in-situ in a relatively stable environment, to create real-time situation awareness. Our experience in operating and maintaining prototype IoT systems in real-world testbeds indicates that integrating mobile devices with in-situ platforms is a promising approach to increase the reliability and sustainability of commonplace community IoT applications. In particular, mobile devices can be leveraged to compensate for the non-uniform availability of infrastructure efficiently. Realizing the potential of the combined "mobile and in-situ" deployments requires us to address a new set of challenges for data collection in dynamic settings.

In this thesis, we propose planning-based approaches to the efficient operation and maintenance of community-scale IoT deployments that consist of both mobile and in-situ devices. Our proposed techniques leverage the prior knowledge of data characteristics, device heterogeneity, community infrastructure, and application needs. The goal is to optimize the activities of the devices under data budgets and timeliness constraints and seek a balance between data utility (i.e., accuracy, importance, and timeliness) and operational cost.

We explore our solution within the context of urban environmental sensing and address three

major research problems regarding IoT data generation, data upload, and sensor calibration (i.e., maintenance), respectively. First, we propose a spatiotemporal scheduling framework that regulates the data generation activities of participating devices. The framework employs online planning algorithms that optimize the spatiotemporal coverage of collected data to meet the application requirements of heterogeneous data types. Second, in the case of non-uniform network availability, we design a two-phase upload planning approach that creates data upload plans (i.e., when, where, and what to upload) for mobile data collectors before their departure (i.e., the static planning phase); the plan can then be adjusted during execution based on the dynamicity they observe (i.e., the dynamic adaptation phase). Finally, to increase accuracy and ensure consistency of collected data during a relatively long period of operation, we propose a multi-sensor calibration planning solution that determines the number of calibration iterations, the time at which they take place, and for each of them, the sensors to calibrate and the number and paths of mobile calibrators.

Together, the proposed techniques provide a comprehensive approach to generate intelligent plans for data collection and sensor maintenance in smart communities that can fully exploit the capabilities of mobile and in-situ devices. We validate our approach in a proof-of-concept IoT system, SCALECycle (based on the SCALE affordable IoT solution for communities), and conduct measurement studies at the community scale.

Chapter 1

Introduction

In this chapter, we introduce the Internet of Things (IoT) and provide a summary of the major characteristics of community IoT deployments. Such large-scale IoT systems and applications provide near-real-time situation awareness to the communities, which in turn requires the collection and delivery of data from heterogeneous devices through heterogeneous networks as the fundamental services. Through a use case in urban environmental monitoring that leverages both mobile and in-situ IoT deployments in communities, we illustrate the key data collection challenges and describe the efforts in this thesis towards addressing these problems. In particular, we strive to fully exploit the potentials of the mobile plus in-situ IoT deployments in community IoT systems by appropriately leveraging community inputs (e.g. device location, network coverage, and physical structure) and application requirements (e.g. spatiotemporal resolution, importance, timeliness, and data accuracy) in the intelligent planning of device activities.

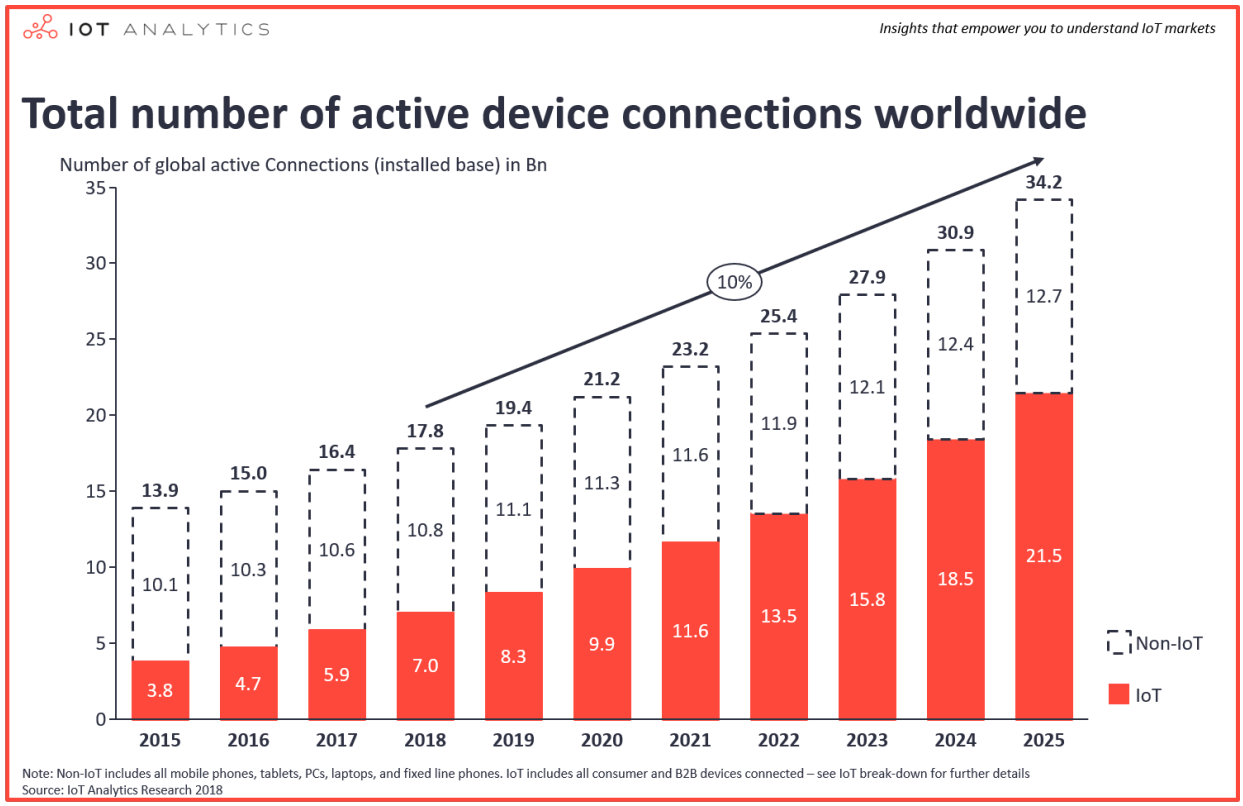


Figure 1.1: Statistics in the 2018 market report from IoT Analytics, showing the increasing number of IoT devices comparing to the total number of global connected devices.

1.1 Internet of Things (IoT) in Communities

The Internet of things (IoT) is an extension of Internet connectivity into physical devices and everyday objects. Embedded with computing and networking devices, these objects can communicate with each other and with numerous services that are available on the Internet. Compared to the earlier techniques to build interconnections between physical devices (e.g., cyber-physical systems, industrial communication networks, etc.), IoT leverages the interconnectivity as a key architectural principle. Today, we heavily utilize the Internet to ensure its low cost and complexity by offloading a significant portion of its processing logic to the cloud, where diverse IoT applications are supported by the comprehensive analyses of the data from numerous devices.

IoT has experienced rapid growth in the last decade. According to the 2018 market report

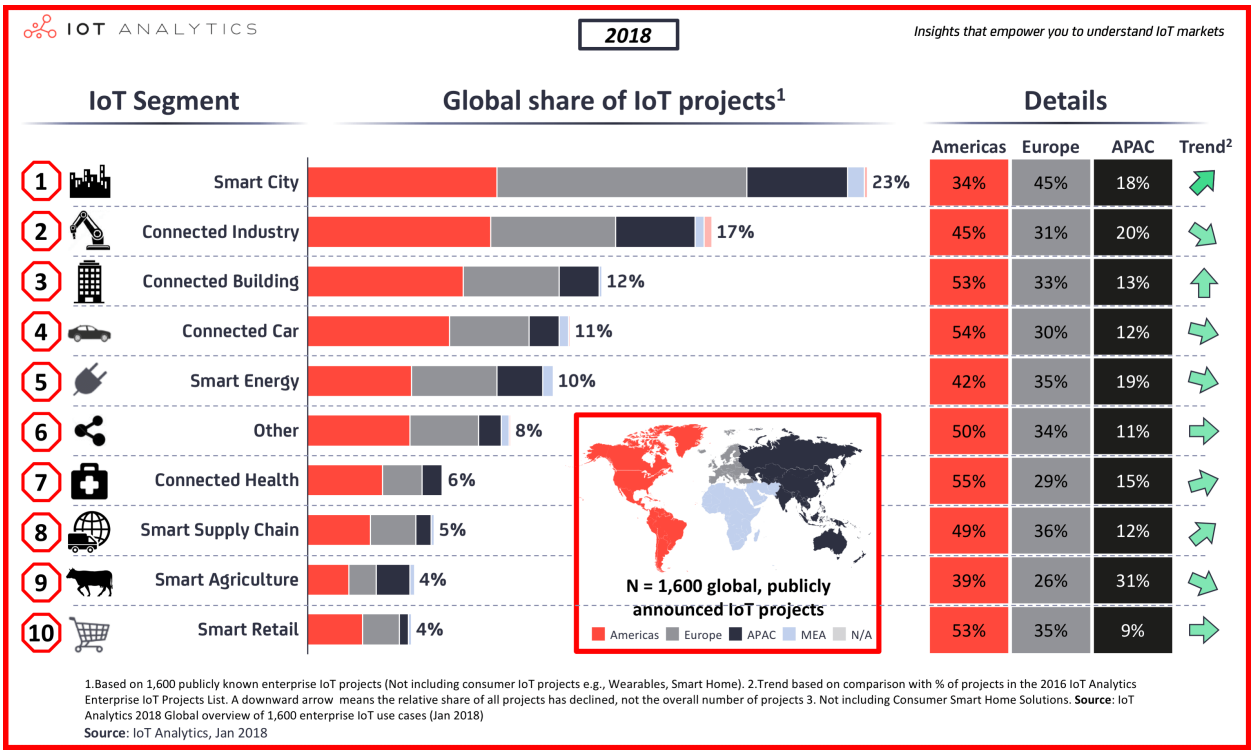


Figure 1.2: Smart city is the top IoT implementation, accounting for 23% of 1,600 publicly known enterprise IoT projects worldwide.

from IoT Analytics [62], there were 7 billion IoT devices, accounting for 41% of the 17 billion global connected devices. Note that in 2015 there were only 3.8 billion (27%) of the 13.9 billion total devices, representing an increase of 84% in three years (Figure 1.1). It is projected that IoT devices will outnumber non-IoT devices by the year 2021. Meanwhile, the global IoT market (i.e., total end-user spending on IoT solutions) has reached 151 billion dollars, a 37% growth from 2017.

With the emerging popularity of smart devices, the scale and form of IoT deployments exhibit significant diversity – use cases range from personal sensing with wearable electronics and smart home devices, to creating environmental awareness in smart instrumented communities and cities [131, 49, 87, 98]. Today, designing IoT platforms and applications to enable pervasive computing in smart and connected communities and cities has become a hot topic. By 2018, **smart city** has become the top category of IoT implementation, ac-

counting for 23% of 1,600 publicly known enterprise IoT projects worldwide [98] (Figure 1.2). Recent events such as the SmartAmerica Challenge [85], the NIST/US Ignite Global City Teams Challenge [74], and the Smart Cities Challenge (Canada) [79] have brought together teams from governments, universities, and industry to demonstrate how lives of citizens can be improved by providing them with effective approaches to monitor and control their surroundings. Popular domains of community-scale IoT applications include environmental awareness, energy conservation, public/home safety, and emergency response technologies.

Our proof-of-concept smart community project, the Safe Community Awareness and Alerting Network (SCALE) [11, 115], is an example of such smart community efforts. SCALE aims at creating an IoT assisted community awareness system to improve the safety and health of residents. To build a "smart" community with enhanced levels of convenience and safety for individuals, we aim to exploit the heterogeneity of collected data and leverage any and all available sensing modalities for increased sensing coverage in communities. In the second phase of SCALE (a.k.a. SCALE2 [115], an NSF-sponsored research effort), leveraging the SCALE hardware/software architecture as a platform and the SCALE deployments as real-world testbeds, we build many community IoT research projects. SCALECycle is an effort to add mobility into the in-situ SCALE deployments as an enabler of better sensing coverage and lower deployment cost to compensate for the non-uniformly available community infrastructures that cause difficulty for deployment occasionally.

SCALE and SCALECycle depict a promising image of future communities with intelligence provided by pervasive deployments of IoT, while our exploration experience with the projects reveals challenges to address and difficulties to overcome.

1.2 Mobile and In-Situ Deployments

The typical architecture of IoT systems consists of (a) devices that detect the physical phenomena; (b) communication networks that deliver the messages from the devices to the cloud; (c) data exchange services that facilitate high-level communication between other services and applications; (d) applications that conduct comprehensive data analyses and serve various purposes. With the broad picture of IoT systems and applications in mind, we delve into the bottom layers of the architecture – the devices and the communication networks, where data are generated from sensed phenomena, processed, and transmitted out of the community landscape to the cloud. The reliability of these layers is the foundation of all the high-level IoT functionality.

Based on how the devices are placed/deployed, we observe that there are two kinds of IoT deployments: (a) **in-situ deployments** that consist of stationary devices deployed in communities and cites – usually mounted on or connected to existing physical infrastructures (e.g., street lamps, bus stops, and buildings), and (b) **mobile deployments**, where devices are usually carried by people or mounted on moving objects (e.g., vehicles) and capable of computing and sensing when they are moving. This dichotomy reveals differences between the capabilities and limitations of the two types of deployments. Both types of deployments present exciting opportunities but exhibit particular challenges (Table 1.1). In-situ deployments can be easily made with a large number of low-cost devices and have advantages for their (a) lower operation cost (because they typically need little human intervention in everyday operation), and (b) the relatively safe and stable environment, thus find themselves good for continuously running systems that support real-time situation awareness and alerting. Besides, the existing facilities in the communities often provide convenient access to power supplies and communication networks, which makes in-situ deployments a good fit for community IoT systems. In comparison, mobile deployments provide a higher degree of flexibility with a lower dependency on the infrastructure, hence are often used to extend the

Table 1.1: Comparison between in-situ and mobile deployments in community IoT systems.

	In-Situ Deployments	**Mobile Deployments**
Infrastructure Dependency	Able to leverage existing infrastructures to enable real-time monitoring	Lower dependency on power and communication infrastructures; able to visit regions where it is hard to deploy infrastructures
Location and Coverage	Street lamps, signal lights, bus stops, other public facilities and buildings (indoor/outdoor); often limited by availability of infrastructures	Mounted on buses, golf carts, bicycles, and UAVs, or carried by trained workers and resident volunteers; can be used to extend sensing coverage
Environment	Relatively safe, stable, predictable, and controllable	Dynamic and hardly predictable; exposed to uncertainty in communication and mobility
Flexibility	Able to support more complex sensing and computing work if there is stable access to infrastructures	Capable for on-demand deployment
Cost in Deployment, Operation, and Maintenance	Lower operational cost because little human intervention is needed in everyday operation; higher maintenance cost for the large number of devices	Lower cost for using less devices to cover the space; higher cost on individual devices for more complex functionalities and the human labor needed to provide the mobility in the case of on-demand deployment

sensing coverage to places where we find it hard to deploy in-situ devices. Mobile platforms also facilitate on-demand deployments and can be dispatched on-the-fly to needed locations, which is extremely useful in critical events. Mobile sensors are also useful in scenarios with infrequent and specialized sensing needs (e.g., detecting poisonous gases in a fire or a gas leak), where permanent deployments are often expensive. In addition, mobility is a good an natural fit in community setups, given the large number of mobile entities that already exist in the communities – golf carts, trucks, buses, etc.

A realistic community environment with diverse applications requiring heterogeneous inputs can leverage both mobile and in-situ deployments combined, and benefit from their advantages. However, such a combination also exhibit new challenges that necessitate planning techniques. In this thesis, we use urban environmental monitoring as our driving use case and identify key data collection challenges under its context, based on which we propose planning approaches for three stages of the system operation.

1.3 Driving Use Case

To lend focus to the problem development and prototype environment design, we employ a targeted use case – **community-scale urban environmental monitoring**. Environmental monitoring (e.g., temperature, humidity, wind, pollution, noise) has been a hot topic for decades, and it is increasingly crucial in densely populated urban areas.

As an example, awareness of air pollution is of extreme importance – studies [9, 56, 108, 129] point to significant correlation between urban air pollution and mortality, raising public concern [132]. Traditionally, urban air pollution, as well as most of other environmental metrics, is captured by city-owned monitoring stations. While they help us to collect data with higher accuracy, they are expensive to setup, operate, and maintain. Given the limited number of such stations [64], information is often at coarse levels of granularity.

Today, with in-situ and mobile IoT deployments in communities, environmental data can be gathered **at much higher spatiotemporal resolutions** to support analytical and decision-making applications: in-situ devices can be mounted on street lamps, traffic signal lights, bus stops, or in residents' homes, while mobile devices can be mounted on buses, golf carts, and bicycles. There could be additional mobile deployments: in everyday operation, residents of the communities can volunteer to participate in sensing or data collection with

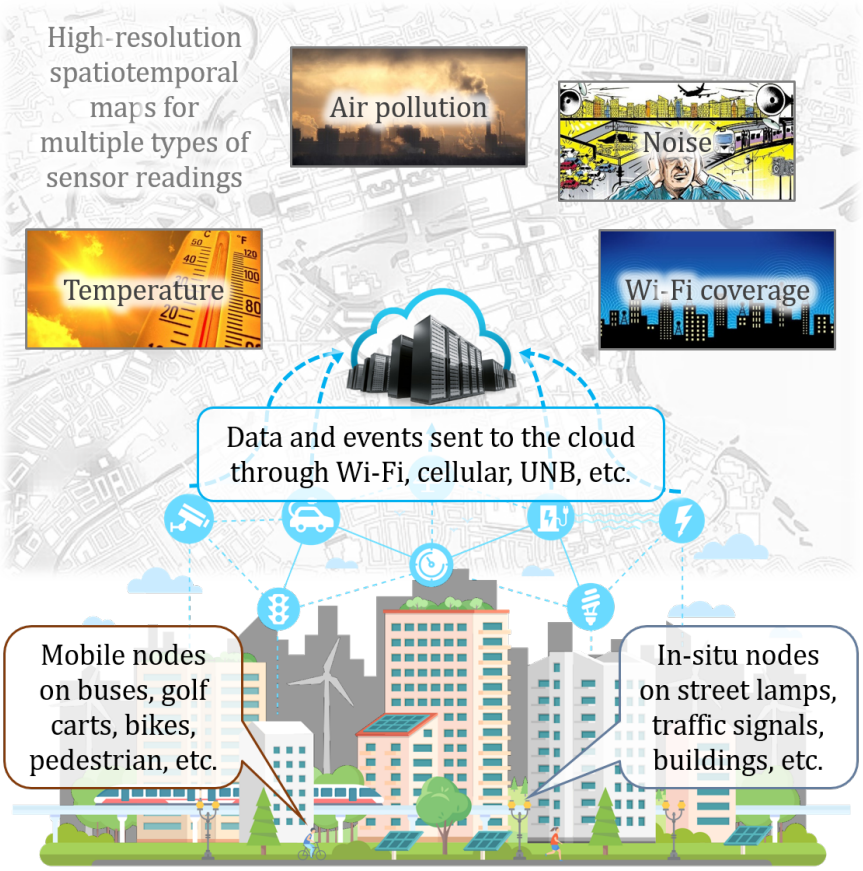


Figure 1.3: The driving use case: In-situ and mobile sensing devices report data through community network infrastructure to the cloud; applications on the cloud analyze the data and create high-resolution spatiotemporal heatmaps for several types of phenomena.

their own mobile devices, while in critical events, groups of first responders can carry mobile devices dedicated to specific tasks. Residents in communities and cities can leverage such IoT applications to plan their travel wisely, e.g., to avoid polluted regions while commuting or find quiet parks with good wireless networks for reading. Decision makers can also benefit from the data and create better urban plans.

As is discussed earlier in §1.2, in-situ IoT deployments are valuable with better connectivity, power supply, and sometimes accuracy, while mobile devices augment the IoT deployment with extended coverage and additional sensing capability. In this thesis, we strive to create a uniform framework that leverages the characteristics of both modalities.

1.4 Key Challenges

With the study of related work in the fields of IoT, mobile computing, DTN, crowdsourcing, etc. and our real-world experience gained from the SCALE and SCALECycle deployments, we identify/encounter several challenges related to community IoT data collection. This thesis aims at addressing the following key challenges: (a) large and unbalanced deployment, (b) non-uniform network connectivity, (c) application diversity and data heterogeneity, and (d) lack of data accuracy and consistency.

Large and Unbalanced Deployment

In the extent of communities, there could be hundreds to thousands of devices serving different purposes. The operation and maintenance of a large-scale deployment require energy, network access, and labor. While the deployment of in-situ devices could be planned ahead, the location of mobile devices is rather dynamic, unbalanced, and sometimes unpredictable (e.g., the devices owned by the crowd). In extreme cases (e.g., an emergency), the crowd might be guided to evacuate from the impacted region, while the first responders are sent into it. Without appropriate planning of device activities and allocation of resources, such large and unbalanced distribution could lead to degradation in system functionality (e.g., data redundancy and network congestion in some areas and lack of sensing coverage in others) and even affect the critical missions. As an example, in the Mendocino Complex Fire (near Lodoga, California) in summer 2018, the Santa Clara fire department reported that Verizon throttled their "unlimited" data plan during firefighting [89]. Considering that firefighting today often leverages multimedia technologies for better situation awareness which add to the communication needs, such failure in resource allocation was accused of "risking harm to public safety".

Non-Uniform Network Connectivity

IoT systems depend heavily on network infrastructures, such as cellular networks, community Wi-Fi networks, and ultra narrowband (UNB) systems. Each type of communication network exhibits unique characteristics, including throughput, latency, range, and cost. At the same time, none of them are likely to cover the community landscape uniformly and continuously. As an example, during our measurement study with SCALECycle, one of our real testbeds was the University of California, Irvine (UCI) campus. We used a Wi-Fi dongle as the "sensor" on the SCALECycle node and collected Wi-Fi RSSI and quality data along the Inner Ring Road (around the Aldrich Park). Heatmaps created from the measurements demonstrate the non-uniform coverage of the campus Wi-Fi system that uses the ESSIDs "UCInet Mobile Access" and "eduroam", given that this system is intended/supposed to provide ubiquitous Internet access to the people on the main campus. At the same time, even in places with good Wi-Fi coverage, the handover process between access points (APs) is still far from being seamless. This might be acceptable for lightweight Internet applications (e.g., messaging), but could be severe for our mobile sensing devices. Actually, during our measurement study, the SCALECycle box on the bike was barely able to send anything through the campus Wi-Fi unless I occasionally stop and let it wait for connection. Additionally, personal experiences show that the cellular networks cover the Aldrich Park and nearby open spaces well, but fail to cover several buildings and structures fully.

Application Diversity and Data Heterogeneity

Earlier efforts in community-scale crowd-sensing and mobile sensing systems usually focus on one application. As computing and sensing technologies develop, more applications can be brought onto the IoT-enabled smart community platform – pollution monitoring, public safety (e.g., surveillance), smart lighting, and smart traffic, to name a few. Running indepen-

dent full-fledged systems for individual applications is of low cost-effectiveness. In contrast, having them on a community IoT platform provides opportunities for (a) data sharing (e.g., presence data could be useful for both public safety and smart lighting, traffic data could be useful for both air pollution monitoring, smart lighting, and smart traffic, etc.), (b) comprehensive data analysis (e.g. learning the correlation between air pollution and traffic, (c) prioritization (e.g., pollution and gas concentration data should gain higher priority near fires or gas leaks). Applications can also have diverse requirements on the spatiotemporal resolution of the collected data. All these factors need to be reflected and accommodated for an effectively coordinated operation of the system. Meanwhile, applications depend on data reported by the deployed devices, while the data exhibit heterogeneous characteristics, including integrity (i.e., chunk size) and pattern of generation (e.g., bursting, streaming, periodic). Heterogeneity also lies in the source – the sensors that generate the data and the devices that could only hold a small subset of all the available sensors.

Lack of Data Accuracy and Consistency

We strive to enable effective IoT data collection, but data are useless without sufficient accuracy and spatiotemporal consistency. The aggregation of relevant knowledge at community-scale from low-cost sensors is problematic since low-cost sensing solutions imply low accuracy and faster degradation/drift. Recent efforts frequently refer to such observations. As an example, in our SCALE experience, all the MQ-family gas sensors have a slightly different response to the same stimuli at the time of instrumentation, and they suffer from significant degradation in sensitivity and zero-point drift at the timescale of weeks to months, forcing periodic maintenance of the devices to calibrate/replace the sensors. Luckily, such relative inaccuracy and inconsistency of the connected devices can be alleviated through the automated calibration of sensors. Mobile devices can have their sensors calibrated at labs/stations. While in-situ devices can have their sensors taken off and brought elsewhere for

calibration in the same way, it is usually more efficient and less interrupting if we send mobile agents to carry accurate reference sensors to calibrate them in-situ. Given that maintenance is always needed for systems to operate in long terms, approaches to reduce the maintenance cost is of high public interest.

1.5 Planning: A Key Driving Principle

The abstraction of network protocols allows most of the smart mobile devices to connect to the community IoT platform without needing to reveal its mobile nature. In particular, many IoT system architecture designs have considered the co-existence of in-situ and mobile devices and strive to provide appropriate infrastructure access to them. However, their advantages (e.g., the flexibility of mobile devices) are often disregarded at the application level. In contrast, we propose to exploit the mobile plus in-situ deployments, focusing on leveraging the mobility with planning techniques to address the application needs. Note that our planning approaches are not limited to planning of the mobility itself (e.g., path planning) – we also regard mobility as part of the context and plan other activities of the devices (e.g., data generation and upload) accordingly.

Specifically, to address the key challenges we discussed in §1.4, we build frameworks to create intelligent plans in three stages of the everyday operation of the community IoT systems: (a) data generation, (b) data upload, and (c) sensor calibration (i.e., maintenance), discussed as follows.

Data Generation Planning

In this stage, the goal is to efficiently ensure the spatiotemporal coverage and utility of the data collected from heterogeneous devices regarding the application requirements. Efficiency

is realized through the planning of sensing (i.e., data generation) activities that aims at reducing unnecessarily redundant data, which ultimately leads to the reduction in storage space, computing and networking resources, and energy consumption. Online scheduling occurs periodically to accommodate the changing location of mobile devices.

Data Upload Planning

In this stage, we focus on how data are uploaded from the devices to the Internet access points (i.e., Wi-Fi APs or IoT gateways with Internet access, which we assume to have a reliable connection to cloud with guaranteed quality of service). The key idea is to avoid time-wasting behaviors in non-uniform network coverage, such as attempting Wi-Fi connection while moving fast or attempting data upload at places with poor network connectivity; at the same time, we optimize for the timely delivery of as many high-priority data chunks as possible. This planning is made possible by leveraging community inputs on the location of good "upload opportunities", as well as the observation made in the field. Complete design of this framework is self-sustaining: the Wi-Fi coverage (captured in RSSI and quality) data and throughput data that are needed for planning can also be collected by the in-situ and mobile devices.

Sensor Calibration Planning

To reduce the overall cost of sending personnel to calibrate deployed sensors in-situ, we organize the calibration tasks of multiple sensors into batches (a.k.a. iterations). In an extended maintenance period (e.g. a few years), to make sure all sensors report data with acceptable accuracy, we find through planning: (a) the number of iterations, (b) the time instants (i.e. days) at which the iterations take place, (c) the sensors that are selected for calibration in each iteration, and (d) the number and paths of one or more mobile agents.

Different sensor types exhibit unique calibration needs (i.e., the time between calibration batches) and calibration cost (i.e., the time needed for calibration). The objective of the planning is to reduce the overall cost that consists of the cost from (a) execution of iterations, (b) calibration, and (c) movement (i.e., traveling).

1.6 Contributions

Overall, in the context of mobile plus in-situ deployments of community IoT systems, this thesis aims to address the key data challenges through planning techniques that exploit the mobility and regulates the activities of devices in three different stages of IoT data collection and maintenance. Key scientific contributions include:

- A holistic community-scale IoT system design that could leverage the advantages of both mobile and in-situ deployments with heterogeneous capabilities while supporting multiple IoT applications with diverse data collection requirements.
- The design and implementation of SCALE and SCALECycle, our community IoT prototype systems, based on which we carry out measurement studies in our real-world testbeds.
- The key data collection challenges in the context of mobile plus in-situ community IoT deployments that we identify from our system experience gained in implementation, measurement study, and maintenance.
- The use of community context and input (e.g., the deployment location of in-situ nodes and the spatial coverage of wireless networks) in comprehensive planning solutions and dynamic adaptation strategies.
- The coordination between continuous real-time data collection provided by dedicated devices and opportunistic data collection enabled by crowd participants.

- The planning approach that enables timely upload of high-priority data in communities with the non-uniform wireless network coverage. The non-uniform nature of infrastructure availability is often oversimplified in related research work.
- The idea of efficient maintenance of a large number of low-cost IoT devices and the planning techniques to achieve such efficiency in the long term.

We believe the combination of mobile and in-situ deployments is the trend in advancing large-scale IoT systems that support smart community/city applications. Intelligent planning of participating devices enables the effective coordination of heterogeneous devices and diverse IoT applications, which could largely improve the overall reliability of IoT systems and the comprehensiveness of IoT-enabled services. This helps create situation awareness, alerting, and adaptive control in communities and ultimately benefits the residents and society.

1.7 Plan of Thesis

The organization of the thesis is listed as follows. For each stage of community IoT data collection, we formalize the respective research problem and provide solution/algorithm to solving it effectively; extensive simulations are based on real and synthetic data, and the results show significant improvement compared to unplanned/naïve approaches.

- Chapter 2 surveys related work.
- Chapter 3 introduces our overall approach to understand and address practical challenges in mobile plus in-situ community IoT deployments. SCALE is our IoT proof-of-concept project, which achieves a high level of flexibility through the functionality abstraction in the SCALE client. We present its architecture, capabilities, client design, and leverage these to create SCALECycle, a mobile data collection platform. Then,

we discuss the measurement study we carried out with and the lessons we learned, with which we identify major data collection challenges, stress on the importance of planning techniques, and derive the research problems in this thesis.

- Chapter 4 describes our spatiotemporal data generation planning framework. It maintains a sufficient level of data accuracy and redundancy through the dynamically planning of the sensing activities on deployed and participating devices. Key contributions include (a) the formalization of the research problem, (b) the design of two online scheduling algorithms, and (c) the extensive simulation driven by realistic community-scale setups and measurements we learned from our testbeds.
- Chapter 5 describes our upload planning approach. It addresses the challenge from non-uniform network coverage and focuses on the timely delivery of important data. Key contributions include (a) the formalization as a constrained optimization problem, (b) the design of a "static-dynamic" two-phase approach and the associated algorithms/policies, which highlights the generality of the Lyapunov-based dynamic adaptation strategies against common sources of dynamicity in communities, and (c) the extensive measurement-driven evaluation that demonstrates the effectiveness of our two-phase approach.
- Chapter 6 describes our multi-sensor calibration planning solution. It improves the long-term efficiency in the calibration/maintenance of heterogeneous sensing devices. Key contributions include (a) the multi-sensor calibration planning as a service that exploits device locality, sensor characteristics, and application needs, (b) the general formalization of the problem and the design of a two-phase iterative solution, (c) the validation that leverages indoor and outdoor settings from our on-going testbeds, and (d) the initial steps towards a prototype of a end-to-end calibration planning service.
- Chapter 7 concludes the thesis with the contributions we made and the lessons we learned, and provides a look forward to the future directions of community IoT systems.

Chapter 2

Related Work

This paper addresses issues at the intersection between IoT systems and mobile computing when deployed at scale in real-world community settings. In this chapter, we survey relevant work to provide an appropriate background for this thesis. We start with an overview of IoT system architecture designs and then discuss how in-situ deployments and mobile devices coexist in community-scale IoT systems today. Next, we explore the related fields of study to understand better how application characteristics and crowd participation are exploited in related fields (i.e., delay-tolerant networking and crowdsensing). Research efforts and techniques in these fields provide deep insights into the development of technologies over the decades, which motivated and inspired our work. Besides, we briefly review the research on sensor calibration and assisting techniques that ease this process, which helps us derive the efficient multi-sensor calibration planning piece.

2.1 Community IoT Architectures

Today, we observe increasing public interest in using IoT technologies in applications at personal and societal levels ranging from personalized healthcare to clean environments.

Recent research in the design and deployment of large scale IoT architectures in community and city-scale deployments have brought new challenges to the forefront. Zanella et al. [131] survey existing and promising technologies in each of the layers in the current Internet protocol stack and provide details of a real-world urban IoT implementation named Padova Smart City. Though not explicitly mentioned, it leverages a typical three-layer IoT system architecture, where the cloud services and the devices use different protocols for communication, and an IoT edge gateway operates between them to enable the compatibility. Perera et al. [87] explain how IoT-based sensing could support smart city applications like waste management, environmental management, and smart agriculture. Jalali et al. [49] focuses on the architecture design of community-scale IoT systems for smart cities to enabled global situation awareness that could benefit residents, service providers, and decision makers. The proposed architecture also contains three layers (i.e., sensing layer, network layer, and services) and targets applications including transportation, healthcare, and public safety.

The above research efforts explore the potential of IoT-enabled smart communities and cities and identify general IoT challenges (e.g., the large number of devices and diverse applications). At the same time, they intend to create general designs and focus on the communication framework and service model rather than particular application scenarios and the characteristics of the end devices. During the system design, mobility is often regarded as a challenge that brings about dynamicity and uncertainty. In contrast, we strive to exploit mobility in the combined setup, leveraging the capabilities of mobile computing to enhance the community IoT systems further.

2.2 Sensing and Data Collection in IoT Systems

IoT-enabled smart community and smart city projects are deployed worldwide. In general, community IoT systems tend to support multiple diverse applications to create comprehensive situation awareness, thus require data from different sources in the service area to be collected and analyzed together – the systems operate as a whole instead of their individual pieces. At the same time, sensing devices lie at the bottom layer of IoT systems. They convert physical phenomena, generate formatted data, and deliver them (actively or passively) to the cloud, serving as the fundamental of all IoT functionalities. Supporting various applications, the IoT systems could depend on in-situ and/or mobile devices for sensing, each showing certain advantages in corresponding application fields.

In-Situ Sensing

In-situ sensing is useful in community IoT systems when we intend to cover a region with a large number of low-cost devices to provide continuous monitoring of specific metrics or events. Sensing devices can connect to community infrastructures for power supply and network access while staying at relatively safe and stable spots. The Community Seismic Network (CSN) [22, 54] is a participatory IoT system created by the California Institute of Technology (CalTech) to help with early alerting of earthquakes in Southern California using cheap accelerometers attached to residents' personal computers and devices. The accelerometers detect changes in acceleration and report such changes as "picks" to the cloud service that runs on Google App Engine. However, a set of picks does not necessarily indicate an earthquake – the cloud service needs to analyze data from all devices in the region to determine whether to generate an alert. The Array of Things [72] is a collaborative effort led by the University of Chicago that leverages multi-purpose sensing devices mounted on buildings, street lamps, etc. to provide real-time, location-based data about the urban

environment, infrastructure, and activity to researchers and the public. It positions itself as a "fitness tracker" for the city, measuring factors that impact livability in cities such as climate, air quality, and noise.

In-situ IoT deployments can also support alerting services. Safe Community Awareness and Alerting Network (SCALE) [11] is an affordable personal and home safety project from our group (DSM) at the University of California, Irvine (UCI). Multi-sensor boxes are placed at residents' homes to provide safety-related sensing capabilities including motion, explosive gas, and personal fall detection. While the community infrastructures facilitate the continuous operation of in-situ deployments, they are not always uniformly available everywhere. They also become the major drawback when they fail, resulting in a resilience crisis in special events or emergencies. During the second phase of the SCALE project (a.k.a. SCALE2 [115]), its real-world deployments serve as the testbed for multiple research projects that aim to improve the resilience of large-scale IoT systems. More details of the SCALE project are presented in §3.

Mobile Sensing

Mobile sensing has been a research topic for years, and now has applications in several domains. Significant advantages of mobile sensing include extended coverage and reduced dependency on infrastructures. Therefore, it has typically been used for data collection in large areas that cannot be blanketed uniformly with sensing devices or network access.

CarTel [46] is a mobile computing system deployed in cars. It supports onboard mobile sensing and data uploading and is used for gathering multiple types of data in cities, including air quality, traffic delay, wireless network coverage, and automotive diagnostics. Communication to the backend is realized opportunistically with the public network infrastructure using Wi-Fi and with data mules (i.e., other CarTel nodes, mobile phones, and removable

storage devices). BikeNet [27] is a sensing platform for mapping cyclists experiences and incorporates both planned operation (a.k.a. tasking) and opportunistic operation. On each BikeNet node, mote sensors mounted on the bike connect to each other using 802.15.4 and form a bike area network (BAN). Communication is through an 802.15.4/Bluetooth module that could opportunistically connect to gateways called sensor access points (SAPs). SAPs could be either modified Wi-Fi access points that are deployed in-situ or mobile phones carried by cyclists with cellular network access (e.g., GPRS). These are early projects that aim to create end-to-end systems from mobile sensing to data visualization. Unfortunately, at the time when these projects were proposed and implemented, public Internet access was still slow and sporadic even in urban areas. Such mobile sensing systems end up with the delay-tolerant networking model and depend on opportunistic communication for data upload, while some of the applications that they intend to support could be time-sensitive (e.g., air quality and traffic delay).

ZebraNet [133, 60] is a mobile sensing network designed for long-term tracking and observation of wild animals. In the initial design of the system, sensing devices are mounted on zebras. Differently from the aforementioned urban sensing applications where devices could get electricity from the vehicles, in the ZebraNet scenario, the availability of both energy and communication is strictly limited. Due to the sparse distribution and low-energy design of nodes, communication is always initiated by dedicated data collectors that wander around periodically. Such a scenario is rare in the everyday operation of community IoT systems, but mobile data collection is still a feasible solution for fetching data from in-situ IoT nodes that are located in certain regions where network access is intermittent, expensive, or damaged.

Recent work on mobile participatory sensing and crowdsourcing has enabled the creation of high-resolution sensing maps for communities and cities using large groups of cheap commodity sensors [107, 38, 21]. Ambiciti [120, 119] (formerly known as SoundCity [86]) is a crowdsensing project from the French Institute for Research in Computer Science and

Automation (Inria). It started with noise sensing using the microphones on participants' smartphones. Based on the collected data, the cloud service creates a real-time noise pollution map for the city of Paris. The real-world deployment of Ambiciti reveals several challenges in the context of large-scale participatory sensing applications, including the consistency of data from heterogeneous devices and the protection of participants' privacy. The project is recently extended to support air pollution sensing, demonstrating the trend of integrating diverse applications in community IoT systems.

Often in such settings, the coverage and deployment of sensing capabilities are tied to the availability of continuous network connectivity [139]. Instead of communication protocols and energy efficiency models, research efforts focus more on the optimal placement of in-situ devices or the path planning of mobile devices [18, 113] in order to achieve a higher spatiotemporal coverage of sensing data. Mosaic [26] is a mobile sensing project from Zhejiang University that uses sensors mounted on city buses to create city-scale fine-grained maps for PM 2.5 (fine particles, an air pollution indicator). To achieve the overall effectiveness, researchers have studied node deployment, sensing coverage [32], multi-hop sensor calibration [29], etc.

Similar techniques are also frequently used in emergency response scenarios to enable on-demand sensing. Raj et al. [95] propose to send trained personnel as mobile data collectors (MDCs) to collect time-sensitive data from certain spots in emergencies. Their work focuses on the path planning of a group of MDCs while new reports of emergencies (i.e., new tasks of data collection) are dynamically added and adapted, assuming that the reports are always possible through specific communication channels.

Indeed, the coverage and throughput of public Internet infrastructure have improved largely in the last decade, making it realistic to assume that devices are always online. However, cellular networks (e.g., LTE) are still expensive for the emerging communication need of rich content, while low-cost, short-range wireless networks (e.g., Wi-Fi) cannot cover urban

regions seamlessly even in highly instrumented facilities like universities – especially if the devices are moving and frequently switching between access points [140]. In community-wide systems, such network availability information could be eventually captured by the devices as in CarTel [46], by leveraging which we could apply more comprehensive techniques to improve the efficiency and timeliness of mobile data collection in real-world.

2.2.1 Exploiting Application Characteristics

IoT data could be time-sensitive or delay-tolerant, depending on the characteristics of the data and the requirements of the applications. Existing work in delay-tolerant networking (DTN) has inspired us to exploit these characteristics to achieve better overall efficiency while reducing the cost and delay of data collection and transmission.

DTN refers to the set of networking approaches that enables data transmission where continuous network access is interrupted or impossible, often due to mobility, dynamicity, or physical limitations. One of the earliest motivation for DTN is the design of the Interplanetary Internet (IPN), where a significant delay and data corruption are inevitable. The concept is later adopted by mobile ad hoc networking (MANET), vehicular ad hoc networking, and sensor networks.

Using mobile devices for message ferrying [16, 134, 135, 67, 15, 123] and data muling [126, 53] has been hot topic under this context. In these scenarios, the applications are assumed to be delay tolerant, i.e., meeting timing deadlines is not a critical goal. Proposed approaches often work with absent or minimal access to communication infrastructure and focus on the route design of ferries/mules or the protocol to enable opportunistic communication. The system design typically involves the use of multi-hop networks [126, 53] and the techniques for proactive and adaptive data transmission/upload [136, 25, 57, 128, 63, 92, 125, 88, 61, 116, 31]. Earlier in this century, the public network infrastructure could not support the

coverage and bandwidth needed for large-scale mobile sensing in communities and cities. At that time, it was common for mobile sensing applications to leverage DTN approaches in their system design (§2.2).

Delay tolerance of applications has also been exploited in the context of mobile crowdsourcing. O2SM (Offline Online Social Media) [136, 25] is a delay tolerant application framework that pre-fetches online social media content when connectivity is available so as to enable efficient offline access to social media streams based on its likelihood of being viewed. O2SM research focuses on the model that is used to infer personal preferences and evaluate this likelihood. Piggyback Crowdsensing (PCS) [57, 128] is a framework designed to reduce the energy overhead of smartphone-based crowdsensing. Using prior knowledge about user habits and CPU/network profiles of applications, PCS determines when to trigger onboard sensors to minimize energy consumption. In this case, data upload only happens at night when the smartphone is connected to Wi-Fi and power adapter at users' homes. In these cases, too, when and how to upload the collected information is not a significant concern.

In this thesis, we aim at the timely delivery of essential sensing data, which is different from the design goal of commonplace DTN scenarios. However, the context-aware planning techniques of DTN inspired the design of our planning approaches, which can be adapted to enhance several application use cases that are potentially time-sensitive.

2.2.2 Exploiting Crowd Participation

Resources and data from crowd users have been leveraged in applications [130], including voting systems, information sharing systems, and social games. In recent years, smartphones with sensors have been used to engage crowd user participation in spatiotemporal dependent tasks. Kanhere [50] points out several challenges in participatory sensing, such as context-awareness and energy conservation. Crowd incentives are also crucial when leveraging crowd

resources, and most studies employ monetary incentives [28, 110]. For example, Feng et al. [28] present an auction framework for the crowd with smartphones in order to maintain truthfulness and individual rationale. Incentives other than monetary have been studied in more recent work [5, 109, 20]. Talasila et al. [109] and Chen et al. [20] propose to leverage mobile and even augmented-reality games to transparently guide mobile gamers to certain places to perform sensing tasks. Our work concentrates on the spatiotemporal scheduling problem and is orthogonal to the aforementioned related work. Liao et al. [59] propose a platform that combines crowd and in-situ sensors for urban sensing. Their work only considers individual tasks at discrete locations; in contrast, our work strives to build complete sensor reading maps in real-time. Zhu et al. [140] propose to leverage node mobility for better coverage and timely data collection in communities. Han et al. [41] formulate a utility maximization framework for mobile crowd sensing that balances data utility and incentive. Khan et al. [52] build a localization framework to estimate the block-level location of participating mobile devices to lower the usage of GPS for energy conservation. Marjovi et al. [64] and Hasenfratz et al. [43] propose to leverage mobile entities in cities to help create high-resolution pollution maps, and focus on using data from a small group of devices and leveraging offline machine learning based techniques to infer the states in uncovered areas. Hachem et al. [38] build a registration middleware for city-scale participatory sensing systems to reduce participation based on the predicted probability of path and capability overlap. Our approach applies to real-time monitoring applications and dynamically selects sensors to activate.

2.3 Long-Term Maintenance of IoT Deployments

The objective of sensor calibration is to find the mapping between inaccurate sensor readings and the "true" values. Therefore, the calibration of an inaccurate sensor often requires the

presence of an accurate sensor (a.k.a. the reference sensor) whose readings are assumed to be the ground truth. Traditionally, the reference sensors are expensive and sensitive; thus, calibration usually takes place in a controlled environment (e.g., labs) [117]. Researchers have to bring back the deployed sensors to calibrate/maintain them periodically, which requires much labor and is especially unfriendly to deployments with a large number of heterogeneous sensing devices (e.g., community-scale IoT systems).

Recent work focuses on automatically calibrating sensors in the field, without relying on a controlled stimulus and without a well-defined range of conditions. CaliBree [66] leverages well-tuned static reference sensors to calibrate low-cost sensors on mobile devices. The standard calibration procedure is to let participating devices (i.e., those holding uncalibrated sensors and reference sensors) communicate and exchange their sensor readings. Upon completion, the calibrated devices can compensate for errors in the software and report calibrated sensor readings directly. However, with the advance of cloud computing, the calibration and compensation procedure can also happen on the cloud as part of the data analytics [101, 42]. Similarly, systems with low-cost static sensors can also be calibrated by mobile reference sensors. For example, [69] proposes to calibrate the sensors deployed on the battlefield with mobile reference sensors.

More advanced calibration techniques have also been recently explored and studied. Blind calibration [7, 105, 124] detects and compensates inaccurate sensor readings using physical characteristics of the sensed phenomena (e.g., distribution of gas concentration), which enables the calibration of in-situ sensor groups without using a reference sensor. The common assumption is that the deployment is sufficiently dense so that the spatiotemporal distribution of the sensed phenomena can be recovered from the samples, even when the actual values are not identical across the region of deployment. Multi-hop calibration [42, 101, 29] allows mobile sensors to get calibrated "indirectly" using other mobile sensors that have been calibrated by the reference sensor. For example, [101] proposes a geometric mean re-

gression (GMR) method for multi-hop sensor calibration. It replaces the traditionally used ordinary least square (OLS) regression and avoids the accumulation of errors across multiple calibration hops. Multi-party calibration [99] derives regression models to solve the calibration problem when there are multiple (i.e., more than two) participants. These techniques improve the level of automation in the sensor calibration procedure and can be leveraged to reduce the workload of humans.

In addition to system architecture and mathematical models, recent work also focuses on planning problems that create opportunities for calibration. For example, the k-hop calibratability study [29] explores the in-situ placement of reference sensors deployed at the bus stops that are used to calibrate the mobile sensors mounted on city buses. Calibration can happen either directly (i.e., between a bus and a bus stop) or indirectly (i.e., multi-hop calibration among buses). The work aims to reduce the number of expensive reference sensor while keeping the number of calibration hops bounded. [69] proposes a TSP-based path planning algorithm for a mobile calibrator. These efforts reduce the complexity of the sensing devices and make calibration activities less opportunistic.

Chapter 3

Approach Overview

In this chapter, we present our overall approach to understand and address realistic challenges in enabling and exploiting mobile plus in-situ deployments in community IoT systems. In particular, to gain realistic expectations of the issues in community IoT deployments and their potential solutions, we have enhanced an earlier Safe Community Awareness and Alerting Network (SCALE) [11, 115] prototype to create SCALECycle, a mobile sensing and data collection platform. We utilize the SCALE and SCALECycle prototypes to conduct measurement studies in real-world testbeds and identify major data collection challenges at two levels: run-time operation and long-term maintenance. We argue that careful planning is required at both levels to address tradeoffs in timeliness, data quality, and cost. In this thesis, we develop novel planning techniques at three stages: data generation, data upload, and sensor calibration.

3.1 SCALE: a Flexible IoT System for Communities

In December 2013, the SmartAmerica Challenge [85] was launched to bring together the industry, academia, and government to demonstrate how IoT and related technologies can bring socioeconomic benefits to communities nationwide. The SCALE [11] project was developed as a response to this challenge by our team at University of California, Irvine (UCI), the government of Montgomery County, Maryland and several industrial partners, aiming to "democratize safety by bringing IoT to everyone". SCALE was a significant technology integration effort that initiated

- Creation of a distributed IoT approach to enabling smart home technologies at a low incremental cost.
- Creation of live testbeds for identifying and researching IoT challenges (e.g. middleware, networking).
- Creation of an open IoT data exchange platform for connecting disparate systems with minimal coordination.

A key design criteria for SCALE is to permit flexibility and inter-operablity of devices, networks, and platforms. In SCALE, sensor data are collected from a variety of physical sensors and processed using "virtual sensors" that extract events relevant to applications. The raw and processed data are communicated through a data exchange platform to interested subscribers (i.e. users, devices, and applications). We deployed the SCALE platform in four real-world testbeds. Among them, the UCI campus deployment has been functional since the beginning of this project and is still reporting data today.

While SCALE was highly successful, it brought out a range of challenges that led to a second phase, SCALE2 [115], which included a set of long-term research efforts that focused on improving the resilience of IoT systems. Here we developed a range of dependability tech-

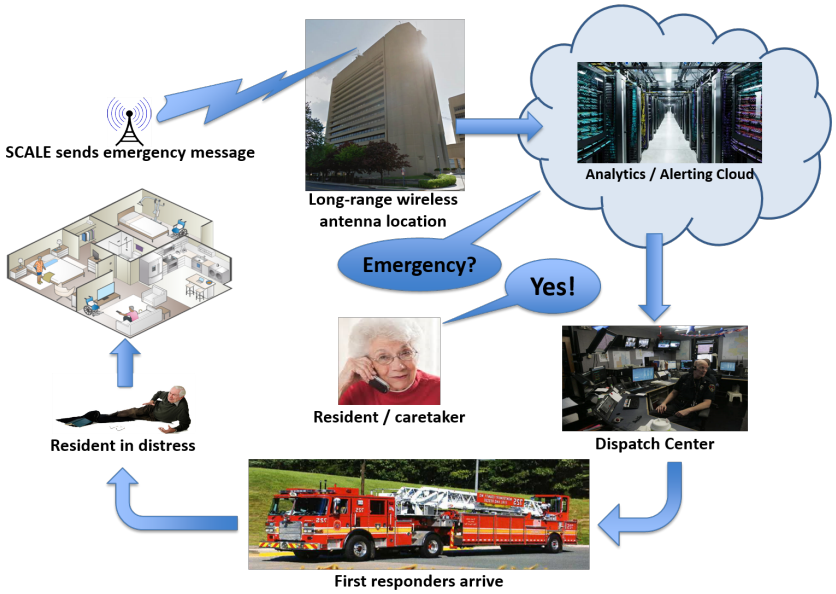


Figure 3.1: The SCALE workflow, using personal emergency detection and response as an example.

niques, including geographically-correlated resilient overlay network (GeoCRON) [13, 12] and resilient IoT data exchange (RIDE) [14]. SCALE was also leveraged as an initial starting point to develop prototype systems in multiple domains, including SCALECycle (§3.2), EnviroSCALE [93], AquaSCALE [40, 39], and SAFER [3, 4]. During the years of assembly, operation, and maintenance of the SCALE devices, and with the measurements we collected from them, we gained experience and insights that motivated the need for exploiting the combination of mobile and in-situ deployments through intelligent planning. Below, we briefly overview the SCALE system's architecture, prototype design, and deployments.

3.1.1 System Architecture

The SCALE system architecture (Figure 3.2) is structured in four layers, bottom up: devices, local network, cloud services, and applications.

In the SCALE workflow (Figure 3.1), physical phenomena (e.g. ground motion and smoke levels) are captured by corresponding sensors on the SCALE devices. The SCALE client

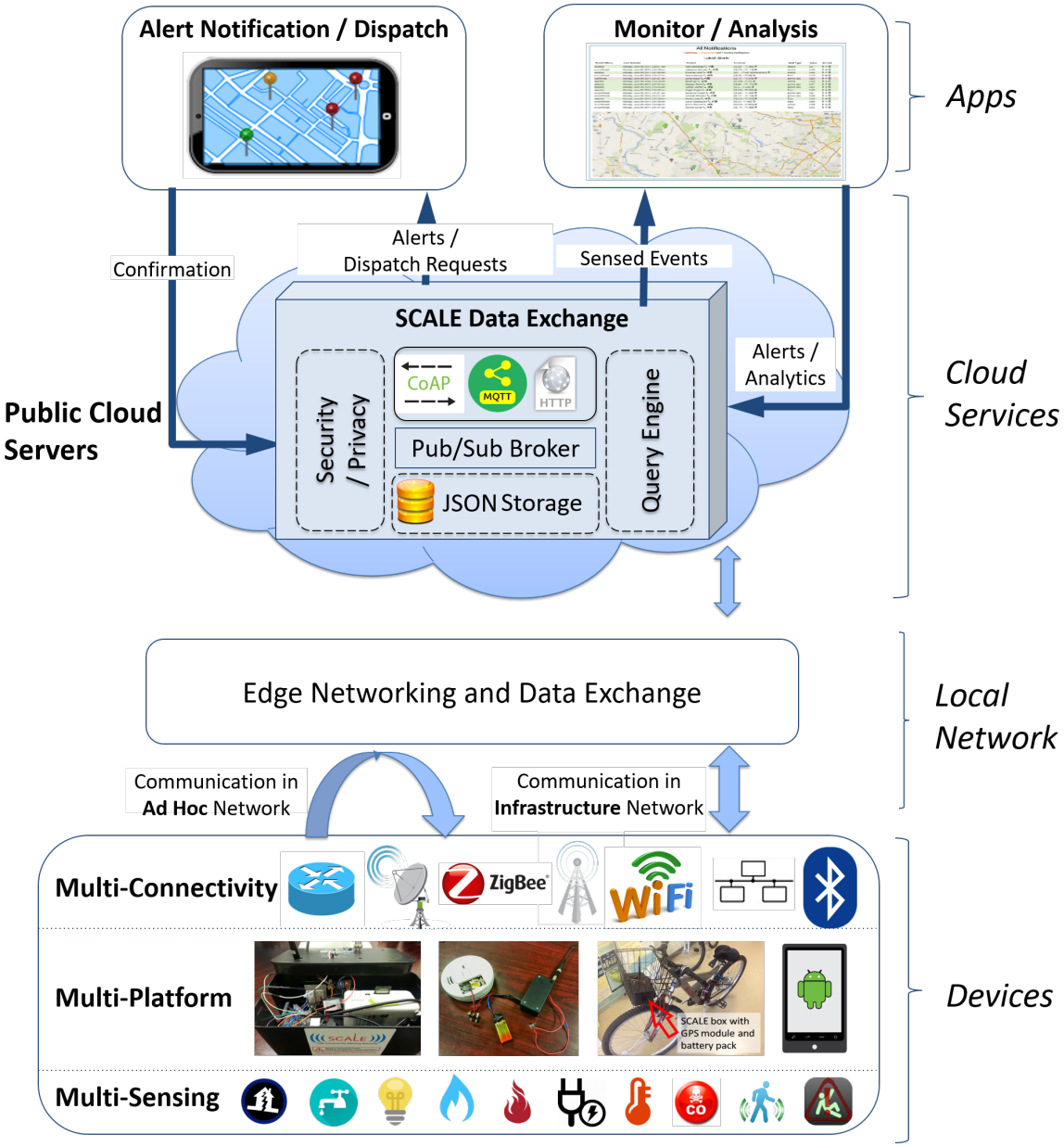


Figure 3.2: The SCALE system architecture, showing major components organized in four layers: devices, local network, cloud services, and applications.

that runs on the devices gets the raw data, conducts initial analytics (e.g. filter, threshold) and generates "sensed events". The events are sent via an internal message queue to the "event sinks" (e.g. publisher) and encapsulated into appropriately formatted messages. The messages leave the end devices and travel through the local network to the cloud services. Applications that are interested in the corresponding types of messages will get them from the data exchange service and leverage them for further analysis and alerting. Our initial use cases were in the domains of personal and public safety, where the final actions have to be carried out by authorities (e.g. first responders), thus are not included in the SCALE system architecture.

In this thesis, we primarily focus on the behavior of devices, where data generation and collection occur. Our planning frameworks run as cloud services and applications, taking into consideration the states of both the devices and the local network, making plans to meet the overall target of timely and efficient delivery of useful and accurate data. Note that much of the state information about the devices and the local network can also be detected by SCALE-compliant end devices and reported in the same workflow. Therefore, SCALE is a self contained and self sustained system that enables our proposed planning techniques.

SCALE Devices: Hardware and Software

The design and implementation of the SCALE multi-sensor box prototype and the SCALE client are among the major contributions of the UCI group as part of the SCALE team.

The SCALE system architecture is general enough to allow the participation of numerous types of devices, from low-profile embedded devices (e.g. Arduino) to general computing platforms (e.g. smartphones and PCs), as long as they interface with our data exchange service. In particular, many of the SCALE partners have their own customized devices. However, having a configurable and flexible hardware platform allows us to further carry out

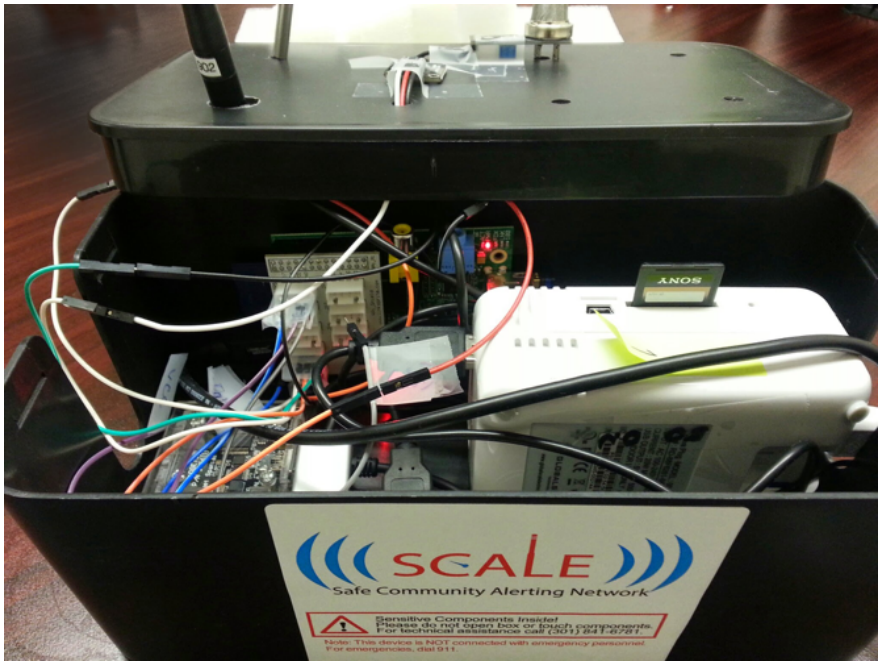


Figure 3.3: A photo of an open SCALE multi-sensor box (2014), showing the SheevaPlug (white) on the right, the Raspberry Pi with the MCP3004 hat (white) in the middle, the SigFox antena (black) on the left, and several sensors.

tests and measurement with heterogeneous sensors, networks, etc. to support our resilience research.

The first SCALE box prototype was created in early 2014, on an ARM-based "plug computer" named SheevaPlug [84]. Due to the limited I/O interfaces on SheevaPlug, the very first prototype supported only USB sensors, for which we had an accelerometer (Phidget) and a temperature sensor. We soon started to use Raspberry Pi Model B to handle commodity analog sensors with the general purpose I/O (GPIO) pins and an MCP3004 analog to digital converter (ADC) hat. The two platforms coexist in the box, as is shown in Figure 3.3, until we finally gave up SheevaPlug in early 2015. The latest SCALE box has a Raspberry Pi 3 Model B, an MCP3008 ADC, a PIR motion sensor (binary output), a USB temperature sensor, an analog light sensor, and several analog gas sensors (e.g. the MQ-family gas sensors). Most of our deployed boxes connect to the Internet via Ethernet or Wi-Fi, while the SigFox (UNB) modules are still supported.

The SCALE boxes, including the initial SheevaPlug-based implementation and the current Raspberry-Pi-based implementation, run embedded Linux. The SCALE client is a daemon written in Python 2 that runs as a Linux service. The client has the following major components:

- An internal message queue implemented by the circuits [73] framework;
- Virtual sensors that map to physical sensors and event detection processes, which generate the sensed events; raw data could be regarded as sensed events, too;
- Event sinks that map to communication protocols (for delivery) or storage space (for caching, logging, etc.);
- An event reporter that implements complex logic that determines for each sensed event which event sink(s) to send to.
- Applications, if provided by users, that implement other functionalities including those that interact with the OS and other hardware/software on the devices.

In particular, sensed events are encapsulated using a standard JSON-formatted data schema we designed, as is shown in Listing 1, when processed by event sinks (e.g. as the payload of an MQTT message).

The SCALE client that runs on our deployed prototype boxes have the necessary virtual sensor abstractions for on-board physical sensors and several event sinks: log, local file, local MySQL database, MQTT, CoAP, and SigFox radio. Each SCALE box can be configured individually and differently using a YAML configuration file.

The manual deployment of a new SCALE box requires (a) the assembly of the hardware, and (b) the installation and configuration of the software (i.e. Raspbian, dependencies, and the SCALE client). According to our testbed and demo-booth deployment experience, this process takes one person and up to four hours of work, which is a reasonably short period.

In large-scale deployments, the software installation phase can be batched with storage (SD card) imaging and flashing, which can further reduce the average time spent on one box to approximately two hours. The SCALE box becomes a perfect fit for our ongoing SCALE2 resilience research, crediting its (a) straightforward assembly and instrumentation process, (b) flexible hardware/software configuration, and (c) highly extensible client architecture. In §3.2, we will describe in detail how we created the SCALECycle platform leveraging the SCALE system architecture and the hardware/software design of devices.

Listing 1 An example of the JSON-formatted sensor data schema that SCALE uses to encapsulate a sensed event.

```
{
  "d": {
    "event": "temperature",
    "value": 27.1,
    "units": "celsius",
    "timestamp": 1385668800,
    "device": {
      "id" : "14c829",
      "type": "raspi",
      "version": "1.0",
      "manufacturer", "chipset", "MAC address", ...
    },
    "location": { "lat": 33.6405, "lon": -117.8443 },
    "cond": {
      "time": 1385668800,
      "value": 27.1,
      "threshold": { "operator": ">", "value": 24 }
    },
    "prio_class": "high",
    "prio_value": 2,
    "schema": { "source": "schema.org/scale_sensors.json" }
  }
}
```

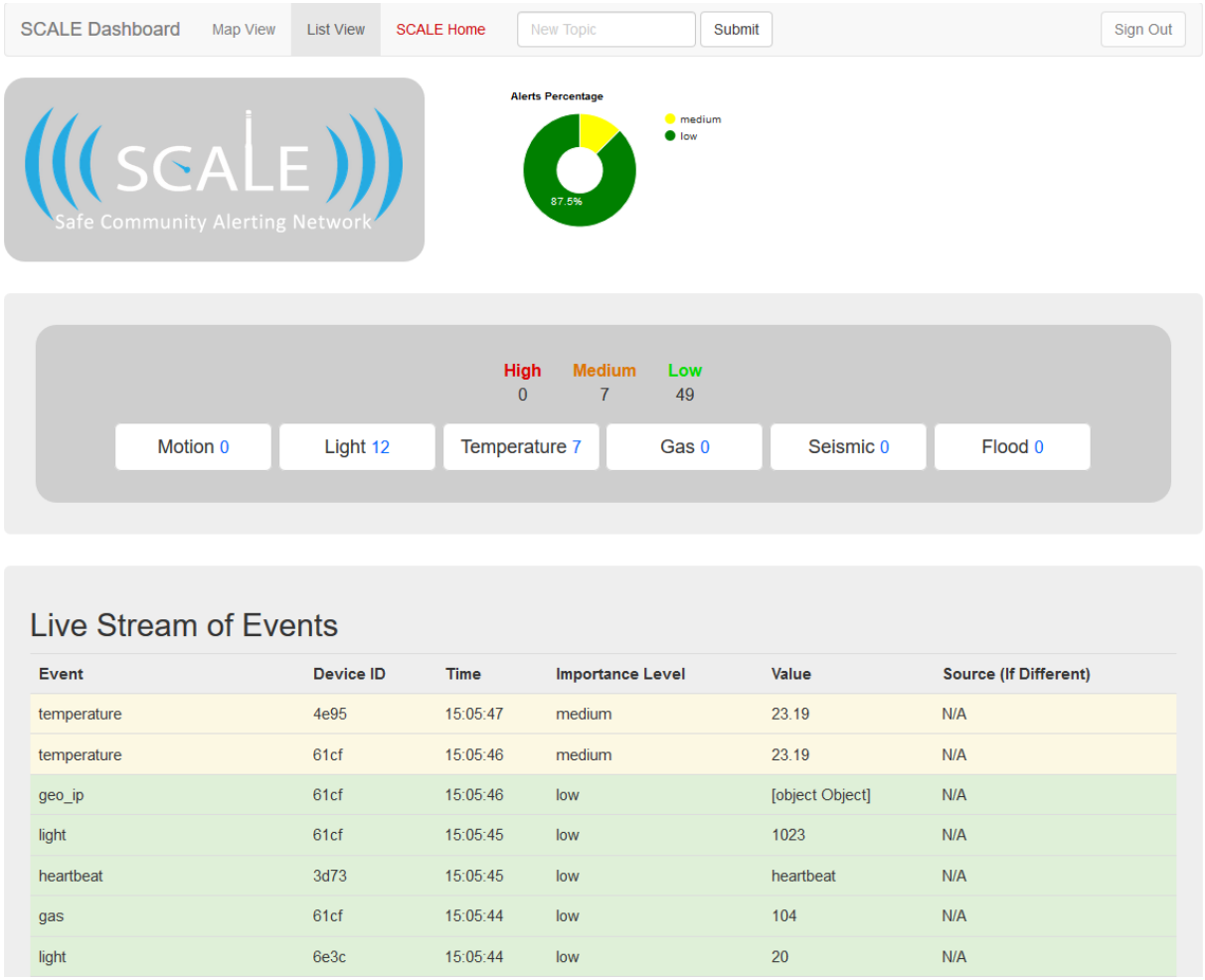


Figure 3.4: The SCALE live dashboard showing a list of sensed events. This is an example application that leverages the SCALE data exchange.

SCALE Data Exchange

IoT deployments must facilitate machine-to-machine (M2M) communication for exchanging IoT data (i.e. sensed events, analytics, alerts). In SCALE, we propose the Data in Motion Exchange (DIME) system. We envision DIME as an open communications hub for IoT that simplifies the development and deployment processes. DIME allows any device or service to publish or subscribe to any other data feed, regardless of the protocols used at the device level. As an example, Figure 3.4 shows a live dashboard application to visualize sensed events and relevant statistics. This simple loose coupling enables developers to incorporate new services and devices without the need to modify existing ones. Any party can introduce new capabilities, or improvements to existing ones, to the system with minimal need for coordination among current components. They can perform analysis on sensed data, or even higher-level events, and contribute the results back to the exchange, driving science and innovation faster as more devices connect.

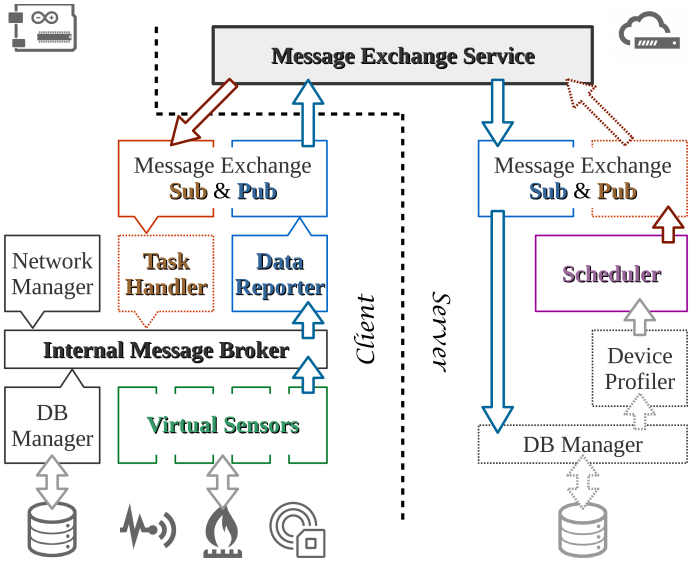
In its current form, DIME uses MQTT [81], a fast, lightweight, publish-subscribe-style protocol. The publish-subscribe model allows multiple servers to collect data from DIME and multiple clients to send it without requiring any configuration on our part. The DIME server currently uses the open source Eclipse Paho MQTT broker. In DIME, sensor data is published to a particular topic, which consists mainly of a device identifier and sensed event type. Other services, such as the SCALE Server, subscribe to this data by a particular device, sensor type, or just to all data. For compatibility, DIME also provides a RESTful interface, implemented via HTTP, initially residing on the SCALE server for ease of deployment. This interface translates incoming data into the proper format and publishes them via MQTT. In this manner, we quickly implemented DIME as a simple MQTT server, though we plan to extend it to directly support other protocols (e.g. HTTP and XMPP).

3.1.2 Real-World Deployments

During the five years of the SCALE project, we deployed the SCALE boxes in multiple testbeds globally, four of which are listed below:

- University of California, Irvine: Devices are on the second floor of the Donald Bren Hall (DBH), where the offices and cubes of our Distributed Systems Middleware (DSM) group are located. We also have a MQTT broker here for simple data exchange;
- Montgomery County, Maryland: Devices are in the Thingstitute IoT Lab located in the Red Brick Courthouse of Rockville and the Victory Court Senior Apartments;
- National Tsing-Hua University, Taiwan: Devices are in the Networking and Multimedia Systems Lab (NMSL) of the Department of Computer Science, featuring collection of multimedia data (e.g. pictures and audio clips);
- Dhaka, Bangladesh: Devices are deployed outdoors, featuring environmental sensing and budget-constrained data exchange using the 3G network (i.e. the EnviroSCALE project).

Currently running applications include (a) a web-based live dashboard showing events sent to the SCALE data exchange service, (b) an InfluxDB-based time-series data storage that keeps raw sensor readings with a simple subscriber program that subscribes to the data exchange service and inserts them to the InfluxDB, and (c) a web GUI (Grafana, open-source project) that visualizes the data stored in the InfluxDB. The SCALE boxes on the UCI campus are recently included in the Testbed for IoT-based Privacy-Preserving PERvasive Spaces (TIPPERS) project [65] as part of the smart building (DBH) testbed.



(a) Architecture diagram of SCALECycle.

(b) UCI prototype.

(c) NTHU prototype.

Figure 3.5: The SCALECycle platform architecture and photos of two prototype devices.

3.2 SCALECycle: Mobile Sensing and Data Collection

The initial SCALE exploration underwent many extensions and expanded to include additional application domains. In the direction of improving flexibility and resilience of SCALE deployments, we designed and built SCALECycle, a mobile sensing and data collection platform. In this section, we present how we created SCALECycle leveraging the SCALE hardware and software. Measurement studies with the SCALECycle prototypes were done on three of the SCALE testbeds.

SCALECycle contains a mobile data collector (MDC), which is a SCALE box augmented with necessary components for mobile sensing and data collection, including different types of sensors and communication interfaces, location tracking units, user interfaces, and power supplies. It can be mounted on bicycles, vehicles, or carried with backpacks to conduct sensing in communities on the go. As the mobile extension to SCALE, SCALECycle helps with extending the coverage of sensing capabilities, responding to instant queries, and collecting data from in-situ sensors when the network infrastructure is down. Figure 3.5 shows

its architecture and two prototype implementations we built on the UCI campus and the NTHU campus testbeds, respectively.

3.2.1 SCALECycle Extension to Hardware

Based on the SCALE multi-sensor box, the major SCALECycle hardware additions include:

- A TGS 2600 analog air contaminant sensor;
- A mini Wi-Fi adapter (Edimax EW-7811 Un) that enables Wi-Fi connectivity and serves as the Wi-Fi RSSI and quality sensor;
- A mini Bluetooth 4.0 adapter;
- A GPS module connected through either Bluetooth or USB;
- A USB portal battery pack (19,000 mAh);
- An Android phone (optional, primarily for debugging purpose).

3.2.2 SCALECycle Extension to Software

The SCALECycle box runs a modified version of the SCALE client to realize the SCALECycle platform design. Most of these modifications have been generalized and merged back to the main SCALE client repository. The major changes in the client software include new virtual sensors that fit better in the mobile sensing context and additional applications/modules that facilitate localization, network management, etc.

New Virtual Sensors

Several new virtual sensors are added to operate the new hardware pieces (e.g. the aforementioned new physical sensors) or provide support for other modules.

- An air contaminant virtual sensor; it inherits the analog virtual sensor class and functions similarly to the light sensor and gas sensor in the original SCALE box;
- A Wi-Fi coverage virtual sensor based on the Python "iwlib"; it reports RSSI/quality data per ESSID (i.e. network name) periodically;
- A network availability virtual sensor based on the "ping" command; it detect if the box has Internet access;
- An upload bandwidth virtual sensor that tests the data uploading rate to the data exchange service.

Location Management

In comparison to in-situ sensing where the location of devices and sensors are predetermined and do not change often, in the mobile sensing context, every piece of sensor data must be geo-tagged to become meaningful. Therefore, location management is critical for mobile sensing platforms, thus we added the following components to the SCALECycle client:

- A GPS location virtual sensor based on the "gpsd" daemon and library;
- A GPS geo-fence application that allows the setup of geo-fences and triggers geo-fence events when a coordinate change reported by the GPS virtual sensor crosses the fence;
- A location manager application that manages the recent location changes of devices and tags sensed events with coordinates.

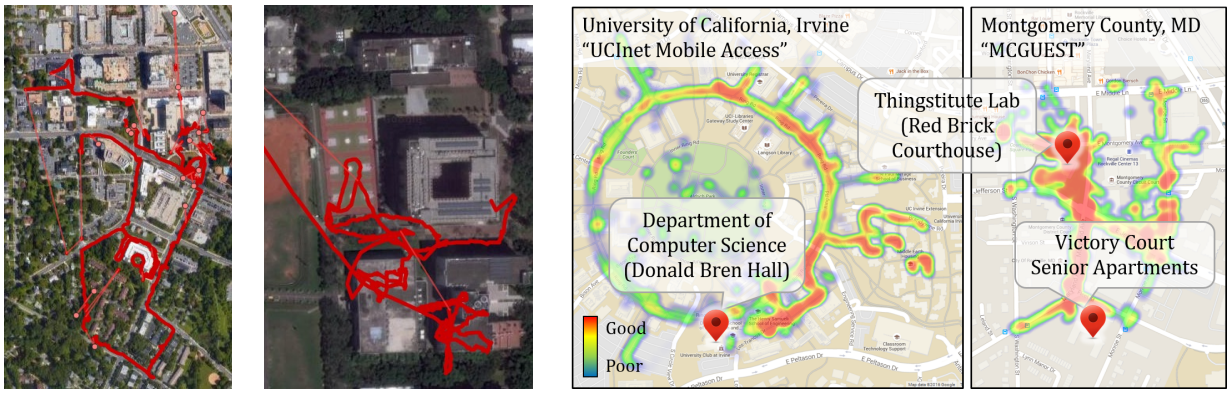
- Additional functionalities in the event/data reporter core component that geo-tag sensed events.

Additional Applications

We also added the following additional applications that are necessary in the mobile sensing scenario.

- A network manager application that monitors the state of network interfaces;
- A local database (DB) manager application and event sink for caching unsent data while disconnected from the Internet and re-sending them once the device is back online;
- A task manager/handler that allows the definition of a series of tasks that can be completed through triggering a specific type of events (e.g. entered a certain zone, connected to the Internet);
- An RF listener application and event sink for receiving commands from and posting instructions to a Bluetooth terminal (e.g. an app that runs on an Android phone), which is often used for monitoring and debugging.
- Additional functionalities in the event/data reporter core component that enable conditional message dispatching to the aforementioned event sinks.

The schedulers that implement the scheduling algorithms run as cloud services and applications (i.e. the server side in Figure 3.5a).



(a) (b) (c)

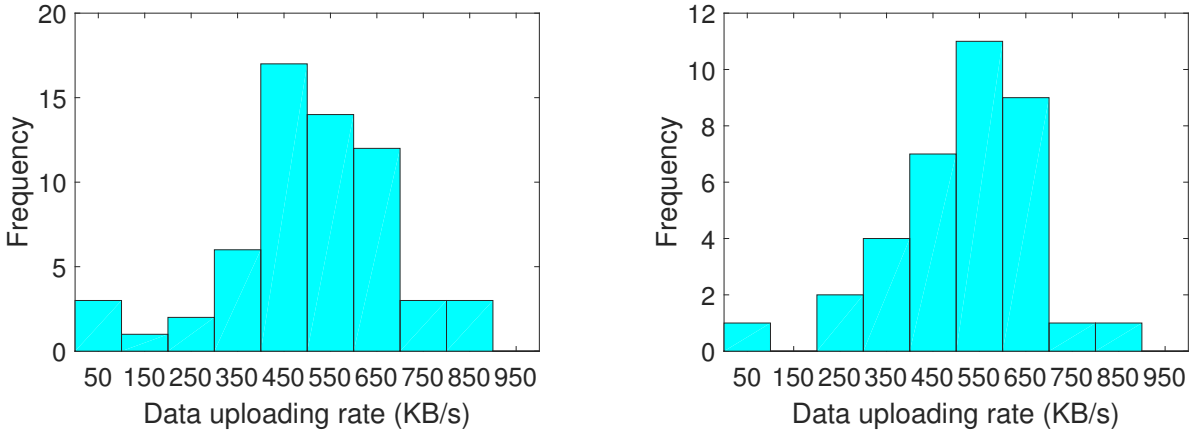
Figure 3.6: Visualization of the measurement we collected with the SCALECycle prototypes. Maps show (a) a GPS path (walking) on and near the Montgomery County testbed, (b) a GPS path (cycling) on the NTHU campus, and (c) two Wi-Fi RSSI heatmaps using data collected from the UCI campus (left) and the Montgomery County testbed (right).

3.2.3 Measurement Study with SCALECycle

We conducted several iterations of measurement study using the prototype systems we created. We collected mobility patterns, Wi-Fi RSSI/quality data, and air quality data on three of our SCALE testbeds: the UCI campus, the Montgomery County testbed, and the NTHU campus. On the UCI and the Montgomery County testbeds, the device is equipped with a TGS 2600 air contaminant sensor and uses its Wi-Fi adapter to collect RSSI data for nearby Wi-Fi APs. On the NTHU testbed, one profile has four MQ sensors and continuous collection of air pollution data at a sampling rate of 0.2 Hz. Another one has a USB microphone and a CSI camera module. During data collection, the SCALECycle boxes were either mounted on the bike or carried in a backpack. To avoid data loss caused by the intermittent Wi-Fi connectivity, an additional local database event sink always saved all sensed events in a local MySQL database on the device itself (i.e. on the SD card). We built a PHP-based visualization tool to plot the measurements on Google Maps as paths (for mobility) and heatmaps (for air quality and Wi-Fi coverage). Figure 3.6 shows the visualization of some mobility and Wi-Fi RSSI measurements we collected.

Table 3.1: Data generation pattern used in measurement study.

Sensor Type	Format	Sampling Pattern	Sample Size
Gas (Analog)	JSON	1 message every 5 sec	200
Microphone	WAV	Clip of 8 sec every min	800k
Camera	JPEG	1 picture every 20 sec	180k
Wi-Fi (iw)	JSON	List of 20 items every 4 sec	3.6k



(a) Montgomery County testbed.

(b) UCI campus testbed.

Figure 3.7: Initial measurements for Wi-Fi upstream bandwidth.

Data generation patterns we observed are listed in Table 3.1. The sample size shown in the table is for uncompressed raw data. In our scenario, analog sensors are used for air pollution related applications, thus they generate data in basically the same pattern regardless of the actual pollution type. Note that data must be sent with enough metadata to identify their source and type. Different application/data types exhibit unique patterns and rates of data generation, suggesting the need for data heterogeneity in our model.

The Wi-Fi upstream bandwidth (Figure 3.7) was around 500 KB/s in all testbeds, with a standard deviation of 100–200 KB/s (testbed dependent). As long as the connection stable, this average upstream bandwidth should be more than enough for the transmission of any or all types data collected for a time window.

3.3 Lessons and Research Challenges

Our experience in the design, implementation, operation, and maintenance of the SCALE and SCALECycle systems has demonstrated the effectiveness of IoT-based community awareness and alerting services. SCALE and SCALECycle, as proof-of-concept research projects, still require further engineering efforts before they can be deployed in large-scale (e.g. to blanket communities and cities). However, we had interesting observations and learned important lessons during the span of the projects through the maintenance work and measurement study (§3.2), which helped us identify the key challenges and derive research problems.

3.3.1 Unbalanced Deployment

In spite of the relatively small size of the deployments we have in our real testbeds, we can see the common causes for the unbalanced distribution/deployment of devices.

While the deployment of in-situ devices could be planned ahead, there are several commonplace constraints. For indoor devices, we tend to place devices at spots where we have stable access to the power supply and the Internet. Privacy issues also prevent the full coverage of certain sensitive types of measurements in both public and private spaces. For example, in the UCI campus SCALE deployments, most of our boxes are deployed in our lab and a conference room. In the TIPPERS [65] project, most of our cameras and beacons are on the second floor of the DBH building, where most of our people's offices are located. We also have our own Wi-Fi systems for our devices. For outdoor deployments, we have additional physical limitations and concerns, such as terrain, weather, and security. In many outdoor IoT settings (e.g. the Array of Things [72]), people tend to choose street lamps, traffic signal lights, etc. for better access to power, and need weather-proof designs like the domes. Not all the constraints are unsolvable, but they could still lead to higher cost of deployment, op-

eration, and maintenance, which naturally leads to the avoidance of such choices [18, 113]. However, the application requirements for data collection are independent from the difficulty. In a park where there is no power supply, Internet coverage, or shelter from the weather, the temperature and air quality data are still useful for people who want to enjoy the outdoor refreshment there.

The location of mobile devices are rather dynamic and sometimes unpredictable (e.g. the devices owned by the crowd). With higher flexibility and less dependency to the infrastructure, mobile devices still cannot go everywhere (e.g. we probably should not ride bikes on the lawns in the park). The mobility of device may also depend on unpredictable events. In critical situations (e.g. an emergency), the crowd might be guided to evacuate from the impacted region, while the first responders are sent into it.

In the extent of a highly instrumented community, there could be hundreds to thousands of devices serving different purposes, resulting in a large and unbalanced deployment. Without appropriate planning of device activities and allocation of resources, this could lead to degradation in system functionality (e.g. data redundancy and network congestion in some areas and lack of sensing coverage in others) and even affect the critical missions.

3.3.2 Non-Uniform Connectivity

IoT systems depend heavily on network infrastructures, such as cellular networks, community Wi-Fi networks, and ultra narrowband (UNB) systems. Each type of the communication networks exhibits unique characteristics including throughput, latency, range, and cost. Wi-Fi networks are short-ranged (e.g. under 50 meters for 802.11n), but existing access points in communities can be leveraged with a low marginal cost. Cellular networks (e.g. LTE and 3G) charge on data transmission, thus are more expensive and require better planning and more efficient techniques [93, 94]. UNB systems are less prevalent than the others and have

several different standards (e.g. SigFox, LoRa), each requiring specific hardware and driver program. Also, due to the low bandwidth, the Internet stack does not work with UNB, so we have to implement our own protocols. As an example, some SCALE boxes have the SigFox radios, but the standard payload is only 12-byte long and it takes approximately 7 sec to transmit it. Hence, the SCALE message schema in Listing 1 did not work. We had to design a compact binary representation for SCALE sensed events and use the radio in emergencies only.

At the same time, none of them are likely to cover the community landscape uniformly and continuously. As an example, during our measurement study with SCALECycle, one of our real testbeds was the UCI campus (Figure 3.6). We used a Wi-Fi dongle as the "sensor" on the SCALECycle node and collected Wi-Fi RSSI and quality data along the Inner Ring Road (around the Aldrich Park). Heatmaps (Figure 3.6c) created from the measurements reveal the non-uniform coverage of the campus Wi-Fi system that uses the ESSIDs "UCInet Mobile Access" and "eduroam", given that this system is intended/supposed to provide ubiquitous Internet access to the people on the main campus. At the same time, even in places with good Wi-Fi coverage, the handover process between access points (APs) is still far from being seamless. This might be acceptable for lightweight Internet applications (e.g. messaging), but could be serious for our mobile sensing devices. Actually, during our measurement study, the SCALECycle box on the bike was barely able to send anything through the campus Wi-Fi unless I occasionally stop and let it wait for connection. Additionally, personal experiences show that the cellular networks cover the Aldrich Park and nearby open spaces well, but fail to fully cover the interior of several buildings and structures.

Indeed, the networking technologies have improved significantly during the last decade, making it possible to deploy a large number of devices in communities and homes. However, the existing technologies are still not ready for ubiquitous coverage and seamless transition. With an appropriate planning framework, we could benefit from the diverse multi-network

and circumvent the limitations.

3.3.3 Heterogeneity

Earlier efforts in community-scale crowd-sensing and mobile sensing systems usually focus on one application. As computing and sensing technologies develop, more applications can be brought onto the IoT-enabled smart community platform – pollution monitoring, public safety (e.g. surveillance), smart lighting, and smart traffic to name a few. Running independent full-fledged systems for individual applications is of low cost effectiveness. In contrast, having them on a community IoT platform provides opportunities for (a) data sharing (e.g. presence data could be useful for both public safety and smart lighting, traffic data could be useful for both air pollution monitoring, smart lighting, and smart traffic, etc.), (b) comprehensive data analysis (e.g. learning the correlation between air pollution and traffic, (c) prioritization (e.g. pollution and gas concentration data should gain higher priority near fires or gas leaks). Applications can also have diverse requirements on the spatiotemporal resolution of the collected data. All these factors need to be reflected and accommodated for an effectively coordinated operation of the system.

Meanwhile, applications depend on data reported by the deployed devices, while the data exhibit heterogeneous characteristics including integrity (i.e. chunk size) and pattern of generation (e.g. bursting, streaming, periodic). During the SCALECycle measurements on the UCI campus and the NTHU campus, we assembled different boxes on the two testbeds. As expected, the raw readings from the sensors encapsulated in JSON-formatted messages have a much lower data generate and smaller chunk size than the audio clips and pictures taken by the microphone and the camera.

Heterogeneity also lies in the source – the sensors that generate the data, and the devices that could only hold a small subset of all the available sensors. For example, a crowd-owned

smartphone participating in noise sensing may not be able to provide humidity and air quality data.

We believe that any planning scheme that attempts to coordinate among devices deployed in the communities should appropriately address the heterogeneity of all sorts mentioned above.

3.3.4 Limited Reliability of Low-Cost Devices

We strive to enable effective IoT data collection, but data are useless without sufficient accuracy and spatiotemporal consistency. The aggregation of relevant knowledge at community-scale from low-cost sensors is problematic since low-cost sensing solutions imply low accuracy and faster degradation/drift. As an example, in our SCALE experience, all the MQ-family gas sensors have slightly different response to the same stimuli at the time of instrumentation, and they suffer from significant degradation in sensitivity and zero-point drift at the timescale of weeks to months, forcing periodic maintenance of the devices to calibrate/replace the sensors. Such difference has also been observed during/after deployment campaigns [8, 70]. Luckily, such relative inaccuracy and inconsistency of the connected devices can be alleviated through the automated calibration of sensors. Mobile devices can have their sensors calibrated at labs/stations. While in-situ devices can have their sensors taken off and brought elsewhere for calibration in the same way, it is usually more efficient and less interrupting if we send mobile agents to carry accurate reference sensors to calibrate them in-situ. Given that maintenance is always needed for systems to operate in long terms, approaches to reduce the maintenance cost is of high public interest.

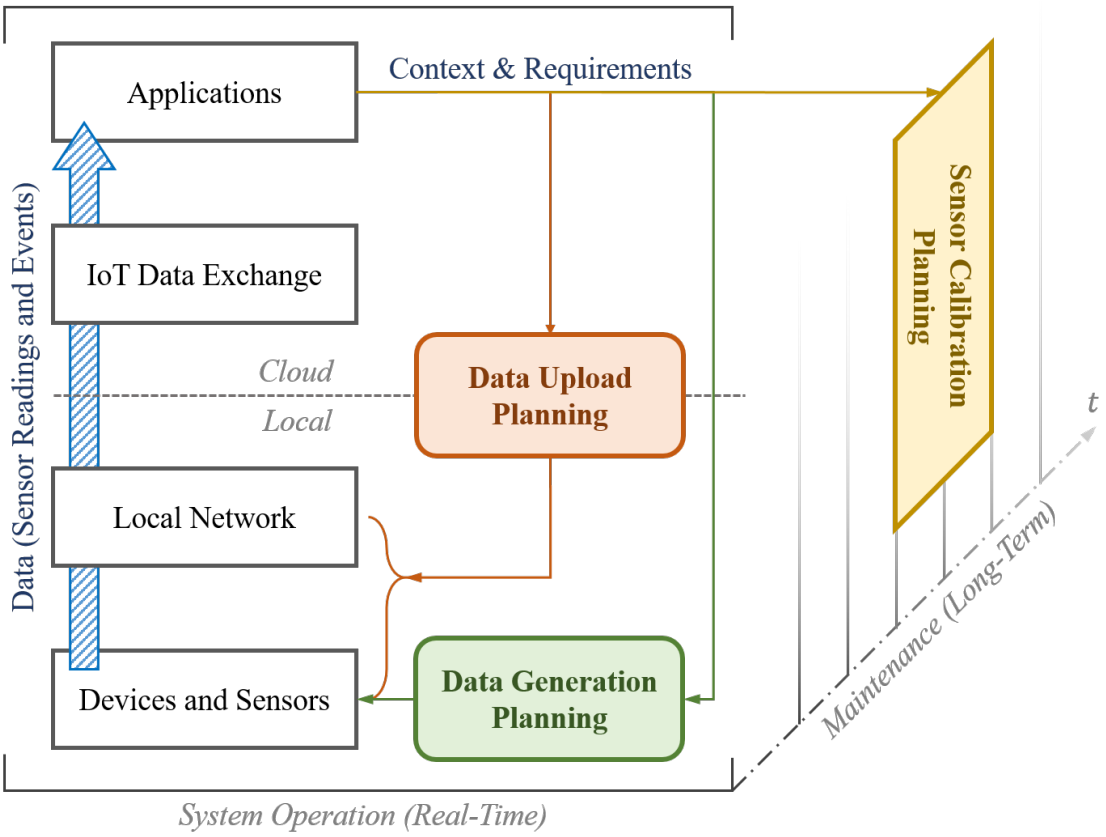


Figure 3.8: The proposed planning workflow for heterogeneous community IoT systems, mapping planning techniques to their corresponding IoT system components.

3.4 Solution Strategy: Phased Planning for Heterogeneous IoT

Seeking to address the major challenges that emerge from exploiting mobile plus in-situ deployment in community IoT settings, we leverage knowledge of mobility with planning techniques to meet application needs. Specifically, we make intelligent plans in three stages of the everyday operation and long-term maintenance of the community IoT systems: data generation, data upload, and sensor calibration (i.e. maintenance) (Figure 3.8). This section provides a brief overview of our approaches. Technical details are expanded in the following chapters.

3.4.1 Spatiotemporal Planning of Data Generation

In this stage, the goal is to efficiently ensure the spatiotemporal coverage and utility of the data collected from heterogeneous devices regarding the application requirements. One promising solution for the management of a hybrid (i.e. in-situ and mobile) environment is to leverage global knowledge (e.g. location and sensing capability of all participating devices) to selectively assign sensing tasks to participants so as to reduce the level of redundancy, while maintaining relatively high accuracy and spatiotemporal resolution of the collected data.

Joint scheduling in such a hybrid configuration requires uniform concepts that capture (a) the diverse spatiotemporal needs of sensing applications and their associated costs, (b) the heterogeneity of devices (sensor types with varying spatial accuracy and compute capabilities), and (c) sensing phenomena that vary in their spatiotemporal extent and dictate the urgency of communication to target recipients. In seek of simplicity, we consider a relatively simple set of applications: our aim (i.e. applications) is to create *high-resolution maps* for multiple commonplace pollution modes (e.g. air, noise, trash), as well as other dynamically evolving phenomena (e.g. traffic, Wi-Fi) in an urban community landscape. Given information on the location and sensing capability of all participating nodes at any time, and the spatiotemporal characteristics of all pollution types, our goal is to determine which sensors on which nodes should be activated during a given time period.

Computing an optimal spatiotemporal plan for activating sensors and devices requires knowledge of all nodal states – this is infeasible given unpredictability of node movements in the future. Hence, we propose an online planning approach, where a cloud service collects information from nodes periodically and generates an activation schedule for the near future. Details of our approach and algorithms will be presented in §4.

3.4.2 Data Upload Planning under Dynamicity

In this stage, we focus on how data are uploaded from the devices to the Internet access points (i.e. Wi-Fi APs or IoT gateways with Internet access, which we assume to have a reliable connection to cloud with guaranteed quality of service). We focus on how to optimally plan the upload of real-time information gathered by mobile devices (a.k.a. mobile data collectors, or MDC) that operate in conditions of intermittently available network contexts, and how to best leverage planning in realistic settings where dynamicity exists in network capacity (i.e. bandwidth), data characteristics (e.g. size, importance, timing), and movements.

We present a two-phase approach to manage data upload for mobile data collectors to enhance the effectiveness of data collection in terms of timeliness, efficiency, and resilience. In the first phase, referred to as **static planning**, a comprehensive plan is computed before the departure of an MDC at the server, based on collected information about the deployment and infrastructures and estimations of parameters. In the second **dynamic adaptation** phase, the static plan is adjusted by the MDC at runtime.

The key idea is to address the dynamically changing networks, data, and movements by exploiting prior knowledge of (a) community networks (e.g. location and quality of Wi-Fi access points or other upload opportunities), (b) the IoT deployments (e.g. where the sensors nodes are), and (c) the heterogeneous nature of IoT applications and associated data characteristics (e.g. volume and modality of data generated during a certain period). Prior knowledge could be learned from daily patterns of mobility and operation, and dynamics could be observed and resolved in real-time. Note also, that much community related IoT traffic (e.g. air quality data) is inherently delay-tolerant to some level. Leveraging any available information and context while being able to adapt to dynamics and uncertainties is the main focus of our proposed upload planning solutions.

3.4.3 Multi-Sensor Calibration Planning for Device Maintenance

So far, we have planning techniques to improve the efficiency and timeliness of data delivery. However, data from sensors lose their utility if the accuracy and spatiotemporal consistency are not guaranteed, which is a realistic drawback of low-cost sensing that is commonly used by community IoT system as we discussed in §3.3. While the relatively small number of mobile sensors can be calibrated at dedicated stations and labs, it could be hard to take back the large number of widely-spread in-situ sensors for maintenance (i.e. uninstall, calibrate, and reinstall) in the same way. To compensate the inaccuracy and inconsistency of deployed in-situ sensors, we send mobile platforms (including humans that carry calibration devices) to travel through the community (i.e. smart spaces), either opportunistically or in a planned manner, to assess the biases of the in-situ sensors and compensate for errors [69]. One can thus generate "sufficiently accurate" knowledge over time through the frequent calibration of the low-cost sensors in the field.

The aim of this stage is to develop a plan for the calibration of a large number of inexpensive (and often inaccurate) sensors using high-integrity reference sensors that are mobile, such that (a) the deployment and operational costs for calibration are minimized while (b) maintaining a sufficient observation accuracy from the sensor measurements. More realistically, given the knowledge of a sensor's degradation characteristics, we plan its calibration with respect to an observed phenomenon so as to maintain an adequate sensing accuracy while minimizing the required effort from the mobile calibrators.

In order to reduce the overall cost of sending personnel to calibrate deployed sensors in-situ, we organize the calibration tasks of multiple sensors into batches (a.k.a. iterations). In a long maintenance period (e.g. a few years), to make sure all sensors report data with acceptable accuracy, we find through planning: (a) the number of iterations, (b) the time instants (i.e. days) at which the iterations take place, (c) the sensors that are selected for calibration

in each iteration, and (d) the number and paths of one or more mobile agents. Different sensor types exhibit unique calibration needs (i.e. time between calibration batches) and calibration cost (i.e. time needed for calibration). The objective of the planning is to reduce the overall cost that consists of the cost from (a) execution of iterations, (b) calibration, and (c) movement (i.e. travelling).

3.5 Summary

In this chapter, we identify major data collection challenges through our system implementation experience gained from the SCALE and SCALECycle projects. We regard planning as the key principle and propose planning approaches to improve mobile plus in-situ community IoT deployments in three stages: data generation, data upload, and sensor calibration. The next three chapters will focus on each of these stages and expand on the details of our models, formulations, algorithms, and experiments.

Chapter 4

Data Generation Planning

We start our exploration of planning techniques from the lowest layer of the system – the generation of the data. In this stage of planning, our goal is to efficiently ensure the spatiotemporal coverage and utility of the data collected from heterogeneous devices to meet the application requirements. Specifically, we address the spatiotemporal scheduling problem to create high-resolution maps (e.g. for pollution sensing) by developing a common framework to capture spatiotemporal impact of multiple sensor types that generate heterogeneous data at different levels of granularity. We develop and validate an online scheduling approach that leverages the knowledge of device location and sensing capability to selectively activate nodes and sensors.

4.1 Chapter Overview

As deployments scale in the number of users and devices, there is increasing redundancy in data traffic, consequently increasing operational cost and resource constraints. Coordination of sensing tasks across multiple devices with diverse sensing capabilities and availability is

essential to the efficient management of the hybrid sensing [30].

One promising solution for the management of a hybrid (i.e. in-situ and mobile) environment is to leverage global knowledge (e.g. location and sensing capability of all participating devices) to selectively assign sensing tasks to participants so as to reduce the level of redundancy, while maintaining relatively high accuracy and spatiotemporal resolution of the collected data.

In this chapter, we address a novel spatiotemporal scheduling problem for crowd augmented urban sensing that supports the monitoring of multiple events in a community. We unify concepts across both in-situ and mobile sensors by defining the notion of *spatiotemporal impact* of sensor readings. We determine how to effectively assign sensing workloads to each participant in order to reduce data redundancy using spatial and temporal knowledge. The key idea is to leverage the knowledge of (a) application requirements (e.g. air pollution), which can be defined by application maintainers and community users; (b) device heterogeneity and mobility (collected at runtime); and (c) spatiotemporal properties of target variables (e.g. concentration of air pollutant), which can be obtained from theoretical models and in-field measurements. Using this knowledge, we propose an online scheduling technique that periodically generates globally optimized plans for all devices.

Consider the environmental monitoring application scenario we described in §1.3. Our aim (i.e. applications) is to create *high-resolution maps* for multiple commonplace pollution modes (e.g. air, noise, trash), as well as other dynamically evolving phenomena (e.g. traffic, Wi-Fi) in a community-wide setting. The pollution monitoring system setup is as follows: Sensor nodes are equipped with varying types of sensors for pollution mapping. Each pollution type (e.g. air, noise, trash) can be characterized using spatiotemporal resolution requirements; sensors for these pollution types have a spatiotemporal impact (i.e. range and duration of sensor data validity) (Figure 4.1). The task at hand is to determine how to control data collection to meet the requirements. A regional edge server is deployed for

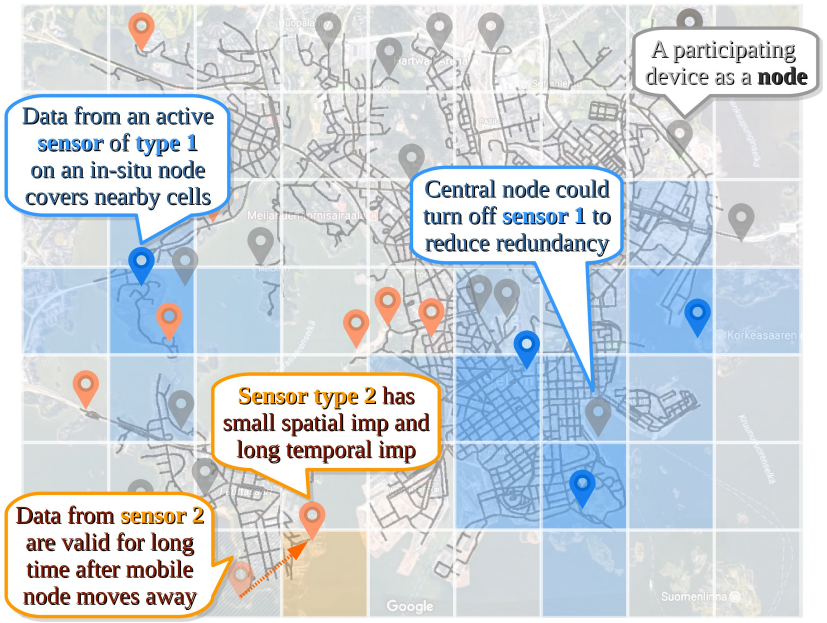


Figure 4.1: An urban crowdsensing scenario, highlighting our approach that selectively activates sensors on participating nodes to reduce redundancy while providing good coverage.

centralized control of devices and simple analyses on raw data. Due to practical bounds on the resources available at the edge server and sensors (e.g. CPU, bandwidth), our plan is to activate a subset of the sensors at any time while retaining the quality of the maps we create, i.e. the sensing activity of each node must be carefully scheduled.

More specifically, given information on the location and sensing capability of all participating nodes at any time, and the spatiotemporal characteristics of all pollution types, our goal is to determine which sensors on which nodes should be activated during a given time period, so that (a) the generated pollution map maximizes the space-time coverage; (b) the total amount of data received by the server is bound; and (c) the total number of workers stays low, so the system suffers from less uncertainty and energy overhead. This spatiotemporal scheduling problem is challenging for the following reasons: (a) Crowd participants move at will; we cannot control their movement or create plans a priori; (b) The varying resolution requirements from applications and the heterogeneous nature of crowd-owned devices make it hard to decouple them during scheduling. In this chapter, we formalize this spatiotemporal

scheduling as a constrained multi-objective optimization problem with discrete space-time representation.

Key Contributions of This Chapter: Computing an optimal spatiotemporal plan for activating sensors and devices requires knowledge of all nodal states – this is infeasible given unpredictability of node movements in the future. Hence, we propose an online planning approach, where a broker (cloud resource or logically centralized edge server in a region) collects information from nodes periodically and generates an activation schedule for the near future. Key contributions of this chapter include:

- Formalization of spatiotemporal scheduling as a constrained multi-objective optimization problem (§4.2), which is NP-hard;
- Design of two online scheduling algorithms (§4.3) that compute sensor activation plans iteratively using states of all nodes and corresponding historical data;
- Extensive evaluation of proposed planning algorithms in realistic simulations driven by the measurements (§4.4) to study performance in community/city-wide settings.

4.2 Spatiotemporal Scheduling in Hybrid Settings

In this section, we define frequently used terms and notations and model the system under appropriate assumptions. Based on the system model, we formulate the spatiotemporal scheduling problem as a multi-objective optimization problem.

4.2.1 Notations and Assumptions

The area of interest (i.e. community/city) is discretized into **cells**. Cells can have arbitrary shapes, but we use square cells for simplicity. Cells are denoted by $c_i, i = 1, \ldots, M$, where

M is the total number of cells in the area of interest. The **spatial distance** between cells is represented by an M-by-M distance matrix $\mathbf{S}$, where S_{i_1,i_2} denotes the distance between the geometric centers of cells c_{i_1} and c_{i_2}.

A **data type** represents a class of sensor data we would like to collect in this area. Data types are written as $d_k, k = 1, \ldots, K$, where K is the total number of data types of interest. We assume each data type requires a different type of sensors for collection. For simplicity, we also assume that we use the same type of sensor for the same data type. Hence there is a **one-to-one mapping**, where d_k also refers to its corresponding **sensor type**. The framework is extensible to complex settings where different sensors types are used for the same data type. Each sensor type has its characteristics on temporal and spatial resolution. The temporal characteristic of d_k is captured by the **temporal impact function** $h_k^T(t) \in [0,1], t \geqslant 0$, which defines the contribution of a data point collected in the same cell, but time t ago. Similarly, the spatial characteristic of d_k is captured by the **spatial impact function** $h_k^S(s) \in [0,1], s \geqslant 0$, for contribution of a data point collected at the same time, but from a different cell that is distance s away. The selection of impact functions is application dependent; the framework should provide interfaces for applications to pass through their impact functions. Typical impact functions have some basic properties, such as $h(0)=1$ (full local impact), $h(x_1) \leqslant h(x_2)$ for $x_1 < x_2$ (monotonicity), and $\lim_{x\to\infty} h(x)=0$. For example, in our environmental sensing scenario, we use exponential (spatial) and threshold-based (temporal) functions. Different types of sensors may generate data at different **rates**. Sensor of type d_k generates data at an average rate r_k. Each data type d_k is given an application-dependent **weight** p_k, s.t. $\sum p_k = 1$.

A **node** represents a participating device. It could be an in-situ sensing platform deployed in the community/city, or a crowd-owned mobile device roaming in the same area. A node is described by $\mathbf{n}_j, j = 1, \ldots, N$, where N is the total number of nodes. We assume we have **no control over the location or movement of nodes**. For consistency, we always use

i, j, and k for indexes of cells, nodes, and data types, respectively. Each node has a subset of sensors **present** on-board. The presence of all sensor types on the nodes is represented by an N-by-K binary presence matrix $\mathbf{B}$, where $B_{j,k}$=1 iff $\mathbf{n}_j$ has the sensor of type d_k. In our hybrid settings, we do not differentiate between in-situ and mobile nodes explicitly in notations. Instead, we focus on node capabilities (i.e. sensors that are present on each node and their impacts).

The **placement** (i.e. location) of nodes is represented by an N-by-M binary placement matrix $\mathbf{G}(t)$, $t \geqslant 0$. $G_{j,i}(t)$=1 iff $\mathbf{n}_j$ is in c_i at time t. We assume that the placement of all the nodes is observable, i.e. $\mathbf{G}(t)$ is known at time t.

We assume each sensor on each node can be **activated** individually at any time. A **plan** describes which sensors on which nodes should be activated at what time. A plan is an N-by-K binary matrix $\mathbf{W}(t)$, where $t \geqslant 0$, $W_{j,k}(t) \leqslant B_{j,k}$, $W_{j,k}(t)$=1 iff d_k on $\mathbf{n}_j$ is active at t. A sensor generates data only when it is active. We say a node $\mathbf{n}_j$ is **active** at t, if at least one sensor on $\mathbf{n}_j$ is active at t.

In real world deployments, planning can occur at any time or when any change occurs; the associated data patterns generated and accumulated can vary. For simplicity, we assume a **discrete representation and operation** in our formulation, where time is discretized into time frames of length T, so that planning only occurs in intervals of $t = n \cdot T, \forall n \in \mathbf{N}$. The activation states of sensors, which is specified in the plan, persist throughout each time frame n. In this way, sensors can only be activated or deactivated per time frame. If an instance of d_k is active in one frame, then it generates $r_k \cdot T$ amount of data for that frame. In the discrete representation, we denote the discrete-time values by $\mathbf{G}[n]$, etc. where $\mathbf{G}[n]$=$\mathbf{G}(n \cdot T)$. We assume $\mathbf{G}(t)$ stays unchanged throughout any frame n.

4.2.2 Definition of Benefit, Cost and Constraints

Benefit: Our goal is to maintain up-to-date heatmaps. We evaluate the plan benefits for the collected data using two perspectives: spatiotemporal **coverage** and data **utility**. Coverage indicates how likely it is that a specific cell c_i has accurate data for data type d_k in time frame n, and utility indicates how useful those data items are, considering redundancy. Since both coverage and utility are closely related to the on/off state of sensors in each cell, we denote the activation state of d_k in c_i using an N-dimensional vector $\boldsymbol{\omega}_{i,k}[n] = [\,\omega_{i,1,k}[n], \ldots, \omega_{i,N,k}[n]\,]$, where $\omega_{i,j,k}[n] = W_{j,k}[n] \cdot G_{j,i}[n]$.

The **single-cell single-frame coverage** $\mathbf{X}^0[n]$ tells whether each cell is *directly* covered by data from at least one node. Its element is represented as

$$x^0_{i,k}[n] = x\big(\boldsymbol{\omega}_{i,k}[n]\big) = 1 - \prod_{j=1}^{N} \Big(1 - \omega_{i,j,k}[n]\Big). \tag{4.1}$$

Even when a cell is not directly covered, it could have **effective coverage** from impacts of historical states and nearby cells. The single-cell single-frame effective coverage matrix $\mathbf{X}[n]$ has elements

$$x_{i,k}[n] = 1 - \prod_{\nu=0}^{n} \prod_{i'=1}^{M} \Big(1 - h^T_k(n-\nu) \cdot h^S_k(S_{i,i'}) \cdot x^0_{i',k}[\nu]\Big).$$

Therefore, the **spatial average coverage** in frame n, $x[n]$, is the average effective coverage over all data types in all cells, and $x[n'{:}n]$ is its average over time frames n' to n, i.e.

$$x[n] = \frac{1}{M} \cdot \sum_{k=1}^{K} \sum_{i=1}^{M} p_k \cdot x_{i,k}[n], \tag{4.2}$$

$$x[n'{:}n] = \frac{1}{n - n' + 1} \cdot \sum_{\nu=n'}^{n} x[\nu]. \tag{4.3}$$

Similar to (4.1), the **single-cell utility** function can be written as $u_{i,k}[n]{=}u\big(\boldsymbol{\omega}_{i,k}[n]\big)$, where the selection of function u is application dependent. A general choice of function u should have these properties: (a) For any data type d_k in any cell, having data collected from more nodes in the cell is more useful than having data from only one node. (b) Having data from many different cells is globally more useful than having multiple data items from the same cell. Replacing letter x in Eq. (4.2) and (4.3), we get similar expressions for $u[n]$ and $u[n'{:}n]$.

Finally, the benefits, i.e. the **overall average coverage** X and the **overall average utility** U are derived respectively as

$$X = x[0{:}z],\;\; U = u[0{:}z], \tag{4.4}$$

which we would like to maximize in our optimization, where z is the total number of time frames during entire operation.

Cost: We depict the cost of a plan using the **number of active nodes**, which reflects the overhead (e.g. core energy and user attention) to keep the nodes active. Scheduling policies can leverage this term to favor the situations where all sensors on some nodes are switched off. It also depends on plan $\mathbf{W}[n]$. According to our definition in §4.2.1, we say $\mathbf{n}_j$ is active in time frame n if $\exists k$ s.t. $W_{j,k}[n] = 1$. Since $\mathbf{W}[n]$ is binary, that is equivalent to

$$y_j[n] = 1 - \prod_{k=1}^{K} \Big(1 - W_{j,k}[n]\Big),\;\; y[n] = \sum_{j=1}^{N} y_j[n], \tag{4.5}$$

where $y_j[n]$ is the single-frame activation state of node $\mathbf{n}_j$ and $y[n]$ is the total number of active nodes in time frame n. The average node activation over multiple time frames $y[n'{:}n]$ is written by replacing the x letter in Eq. (4.3) with y, i.e. the **average number of active nodes** $Y = y[0{:}z]$, which we use as cost to minimize in our optimization.

Constraints: One key constraint that we capture by our definition is $W_{j,k}[n] \leqslant B_{j,k}, \forall j =$

$1, \ldots, N, \forall k = 1, \ldots, K, \forall n \in \mathbf{N}$, which reflects the **hardware configuration**, i.e. no device could activate a sensor that does not exist on it.

The other constraint is the **data quota** determined by the limited server resources and communication infrastructure so that all data could be transferred and processed timely. The total data rate in frame n is the number of active sensors of each type multiplied by the type-specific data rate, i.e. $d[n] = \sum_{k=1}^{K} r_k \cdot \sum_{j=1}^{N} W_{j,k}[n]$, and the average data generation rate through time frames n' to n is $d[n':n] = \sum_{\nu=n'}^{n} d[n]/(n - n' + 1)$. Thus, the **average data rate** $D = d[0:z]$. Note the dimension of D is *byte/s*. With our optimization, we would like to keep D bounded within a predefined data quota D_{quota}.

4.2.3 Problem Formulation

With the assumptions and terms we have, we formulate the spatiotemporal scheduling problem as the following multi-objective optimization problem:

In any time frame $n \in \mathbf{N}$, given the sensor type presence matrix $\mathbf{B}$, the nodal placement matrix $\mathbf{G}[n]$ and the data type characteristics r_k, h_k^T, and h_k^S, $k = 1, \ldots, K$, determine $\mathbf{W}[n]$ that optimizes the expectation of **overall performance** $\mathbf{E}[\Gamma]$, which is defined as the weighted sum of (a) the average coverage X, (b) the average utility U, and (c) the average number of active nodes Y, subject to the hardware configuration and data quota constraints. Formally, this is stated as

$$\begin{aligned} \max_{\mathbf{W}[n]} \mathbf{E}[\Gamma(X, U, Y)] &= \gamma_1 \cdot \mathbf{E}[X] + \gamma_2 \cdot \mathbf{E}[U] - \gamma_3 \cdot \mathbf{E}[Y], \\ \text{s.t. } W_{j,k}[n] &\leqslant B_{j,k}, \forall j = 1, \ldots, N, \forall k = 1, \ldots, K, \\ \mathbf{E}[D] &\leqslant D_{\text{quota}}. \end{aligned} \tag{4.6}$$

This formulation of the spatiotemporal scheduling problem, even when simplified into its single time frame case used in our online approach, is a typical integer programming problem which is known to be NP-hard.

4.3 Algorithms for Online Scheduling

In the absence of a scheduling technique, a simplistic policy is that of complete activation (i.e. activating every available sensor). This naïve "everything" approach may not meet data constraints, but always results in the maximum possible overall coverage and utility with the given inputs and can be used for comparison purposes in evaluation. A brute-force search will compute the optimal solution but obviously only for very small test cases. In this section, we propose 2 algorithms to address the online scheduling problem: (a) an iterative greedy heuristic with improved algorithm termination (§4.3.1), and (b) a Lyapunov control strategy inspired optimization (§4.3.2).

4.3.1 Highest-Score-First Greedy Heuristic

The **highest-score-first (HSF)** greedy heuristic computes a *score* for each sensor on each node (i.e. each node-sensor pair) in a time frame n. It iteratively chooses the node-sensor pair with the highest score for activation, and updates the scores for other pairs, until no selection yields a positive score. The data quota can roll over to the next time frame but cannot be advanced. Its balance in current frame is denoted by D_{q} Algorithm 1 sketches the overall technique.

The **score** $\delta_{j,k}$ of a node-sensor pair (j,k) is defined as the unit-data contribution it makes to the overall performance if we activate (only) this sensor, i.e. $\delta_{j,k} = p_k \cdot (\gamma_1 \cdot \Delta x[n] + \gamma_2 \cdot \Delta u[n] - \gamma_3 \cdot \Delta y[n])/\Delta d[n]$, where $\Delta d[n] = r_k$. The intuition behind HSF is to enhance the

Algorithm 1 Highest-score-first algorithm for finding a plan $\mathbf{W}[n]$ for time frame n, showing the basic procedure and the slow termination phase.

function planHSF ($\mathbf{B}$, $\mathbf{G}[n]$, D_{q}, $\{d\}$, $\mathbf{S}$, $\boldsymbol{\gamma}$, $\mathbf{X}^0$)

Input : Presence $\mathbf{B}$, placement $\mathbf{G}[n]$, and quota D_{q}
Data types $\{d\}$ and their characteristics
Spatial distance $\mathbf{S}$ and weights of objectives $\boldsymbol{\gamma}$
Historical single-cell coverage $\mathbf{X}^0[0:(n-1)]$

Output: Plan $\mathbf{W}[n]$ for time frame n

Initialize $\mathbf{W}[n] \leftarrow \mathbf{0}_{N,K}$; $j_{\mathrm{ST}} \leftarrow$ **null**
$\mathbf{X}^0[n] \leftarrow$ Get single-cell coverage from $\mathbf{W}[0:n]$
cand $\leftarrow \{(j,k) \mid B_{j,k} = 1 \wedge W_{j,k}[n] = 0\}$; sumD $\leftarrow 0$
while *cand* **is not empty do**
    maxScr $\leftarrow 0$; maxPr $\leftarrow$ **null**
    for each (j,k) **in** *cand* **do**
        $\mathbf{W}' \leftarrow \mathbf{W}$; $W'_{j,k} \leftarrow 1$; $\delta_{j,k} \leftarrow$ Get score from $\mathbf{W}[0:(n-1)]$ and $\mathbf{W}'$
        if $\delta j,k > maxScr$ **and** $sumD + r_k \leqslant D_q$ **then**
            maxScr $\leftarrow \delta_{j,k}$; maxPr $\leftarrow (j,k)$
    if *maxPr* **is not null then**
        if j_{ST} **is null then** $W_{\mathrm{maxPr}}[n] \leftarrow 1$;
        else
            $w_{\mathrm{ST},k} \leftarrow 1$; sumScr $\leftarrow$ sumScr $+ \delta_{j,k}$
            if $sumScr > 0$ **then**
                $j_{\mathrm{ST}} \leftarrow$ **null**; $W_{j_{\mathrm{ST}},*} \leftarrow \mathbf{w}_{\mathrm{ST}}$
        sumD $\leftarrow$ sumD $+ r_k$
    else if j_{ST} **is null then**
        $(j,k) \leftarrow$ Get pair for max score without Δy
        $j_{\mathrm{ST}} \leftarrow j$; $\mathbf{w}_{\mathrm{ST}} \leftarrow \mathbf{0}_K$; sumScr $\leftarrow \delta_{j,k}$; maxPr $\leftarrow (j,k)$; sumD $\leftarrow$ sumD $+ r_k$
    if *maxPr* **is not null then** cand**.delete** maxPr ;
    else break ;

overall performance metric $\mathbf{E}[\Gamma]$ in Equation 4.6 using the limited data quota.

HSF starts with an empty matrix $\mathbf{W}[n]$, where all sensors are inactive, i.e. $W_{j,k}[n] = 0, \forall j = 1, \ldots, N, \forall k = 1, \ldots, K$. In every iteration (big loop, Ln 4–21), it computes the score $\delta_{j,k}$ for each (j, k) pair, s.t. $B_{j,k}$=1 and $W_{j,k}[n]$=0 (sensor k exists on node j but not yet activated), and picks the (j, k) that gives the maximum positive $\delta_{j,k}$. To compute the scores of all pairs (inner loop, Ln 6–9), for each pair (j_0, k_0), it creates a temporary plan W', s.t. $W'_{j_0,k_0} = 1$ and $W'_{j,k} = W_{j,k}[n], \forall (j, k) \neq (j_0, k_0)$, computes the objective values of W', and subtract that of $W[n]$ from the result to acquire the score δ_{j_0,k_0}. HSF loops until no positive score is possible, or the data quota is used up.

Complexity: The intuitive implementation of HSF has worst-case time complexity of $O(K^2M^2N^2 + K^2MN^3)$, which can be reduced to $O(KM^2N + KMN^2 + K^2N^2)$ with appropriate optimization.

Early termination of HSF: Occasionally HSF terminates due to non-positive scores; if we select a pair (j_0, k_0) with non-positive score, succeeding selections can be made on the same node to amortize the cost of activating j_0, so all active sensors on j_0 together can make a positive total score. In this case, j_0 is a node that should be included in $W[n]$, but were not because of HSF's early termination.

Therefore, we add a "slow termination" (ST) phase depicted in Algorithm 1, Ln 10–19 to HSF. In **HSF–ST**, when the scheduler meets all non-positive scores, it moves (using Ln 17–19) into the slow termination phase (marked by node j_{ST}). Specifically in this phase, it picks the (j, k) that gives the highest score when not considering the $\Delta y[n]$ term in score computation (Ln 18). Then it iteratively adds other sensors on the same node j, keeping track of the total score of all added sensors on node j (Ln 12–15). There are three possible ways that ST ends: (a) After adding a sensor, the accumulative score turns positive (Ln 14). In this case, it adds all sensors selected during ST into $\mathbf{W}[n]$ (Ln 15) and goes back to the

basic HSF loops. (b) Data quota is used up. (c) All sensors on node j are selected, but the accumulative score is still non-positive. In both (b) and (c), all sensors selected during ST are dropped, and $\mathbf{W}[n]$ is returned immediately (Ln 21).

Complexity: The worst-case time complexity of HSF–ST is the same with basic HSF, i.e. $O(KM^2N + KMN^2 + K^2N^2)$.

4.3.2 Data Budget Handling using Lyapunov Control

In HSF–ST (§4.3.1), we try to optimize the overall performance up to time frame n, without considering future possibilities. However, on occasion, we may want to save data for future use when present benefits are marginal, or advance data quota to seize the opportunity for a significant improvement. The Lyapunov control strategy allows us to dynamically handle data budget while keeping the average usage bounded.

Lyapunov optimization [68] is often applied to systems that evolve over time. It maximizes the temporal average reward while keeping one or more "queues" bounded. The framework defines a Lyapunov function on system states and tries to keep the Lyapunov drift (expected difference between function values at two successive steps), as small as possible, which ultimately ensures the system reaches its goal over time.

In our spatiotemporal scheduling, we define the system state $\phi[n]$ as a queue representing the accumulative data rate, i.e. $Q[n] = n \cdot d[0{:}(n-1)]$, which equals the amount of data that have been generated up to frame $(n-1)$ divided by frame length T. Then, the Lyapunov function is $L(\phi[n]) = (Q[n] - n \cdot D_{\text{quota}})^2/2$, and the Lyapunov drift is $\Delta L(\phi[n]) = \mathbf{E}[L(\phi[n+1]) - L(\phi[n])] \approx (Q[n] - n \cdot D_{\text{quota}}) \cdot d[n]$, where $d[n] = Q[n+1] - Q[n]$. The strategy suggests

we minimize the "drift minus reward" $(\Delta L(\phi[n]) - V \cdot R[n])$, which is to maximize

$$\gamma_1 \cdot x[0{:}\ n] + \gamma_2 \cdot u[0{:}n] - \gamma_3 \cdot y[0{:}n] - \gamma_L \cdot \Delta L(\phi[n]). \tag{4.7}$$

In the actual implementation, we use the same γ values as coefficients for X, U, and Y; we pick a value in the order of 10^{-9} for γ_L. Tuning of γ_L and a few other minor tweaks are needed to establish consistency when the scenario scales.

Complexity: The complexity of Lyapunov control depends on the implementation of the maximizer. In our case, we employ a similar greedy heuristic as is used in Algorithm 1 to maximize Eq. 4.7, using the increment of its value as the score in selection of node-sensor pairs. It has the same complexity with HSF–ST.

4.4 Performance Evaluation

In this section, we describe the simulation environment and experimental design for performance evaluation, and present the results with analyses.

4.4.1 Experimental Environment and Simulation Setup

Earlier in §3.2, we built prototype devices to demonstrate the effectiveness of our system architecture and conduct measurement studies on real-world testbeds. Due to the limited scale of our current deployments, we carry out further evaluation in simulations driven by measurements from our testbeds. Simulations are performed using the Opportunistic Network Environment (ONE) simulator [51] and our custom simulation framework written in R. Data generation rates and patterns are tuned to reflect our real-world measurements; and mobility traces are generated by the ONE simulator using its pedestrian model on the built-

Table 4.1: Basic simulation setup for spatiotemporal data generation planning.

Simulation Parameters		Values
Cells	Number M	63
	Size (Length) l_{cell}/m	50
Nodes	Number of All N	100 *or* 50 [a,b]
	Number of Mobile Nodes N_{mob}	40 *or* 20 [a,b]
	Sensor Presence	See Table 4.2 [b]
	Speed of Mobile Nodes v/(m/s)	$[0.5, 1.5]$ [a]
Data Types	Number K	10
	Characteristics	See Table 4.2
Simulation	Length of Time Frame T/min	1
	Length of Simulation T_{dur}/min	180
	Data Quota D_{quota}/(MB/s)	3.5 [a]
	Weights of Objectives $\boldsymbol{\gamma}$	$(1, 0.4/\ln 2, 1.2)^t$

[a] Subject to change if used as an independent variable.
[b] Setup varies in simulation scenarios.

in Helsinki street map [51]. The performance evaluation component and scheduler interface are implemented by our custom simulator written in R, where we can add algorithms as R functions. Our simulations do not consider communication delays or faults. Experiments were done on the ICS Openlab cluster at UCI, where each node has two (2x) Quad-core Intel Xeon 3.0 GHz CPUs, 32 GB RAM, and runs CentOS 7.3.

The basic experimental setup is shown in Table 4.1. As is assumed in §4.2.1, we use nodal locations at the beginning of frame n as the prediction of $\mathbf{G}[n]$ which stays the same throughout the frame n, i.e. $\mathbf{G}[n] = \mathbf{G}(n{\cdot}T)$. This assumption holds as $v{\cdot}T \ll l_{\text{cell}}$. The simulation framework is extensible to support complex predictors. We define the spatial impact h_k^S for each individual data type using a exponential function $h_k^S(s) = \exp(-s/S_k^0), s \geqslant 0$ with a type-specific constant $S_k^0 \geqslant 0$. We define the temporal impact h_k^T using a step-down function $h_k^T(t) = 1 - \theta(t - T_k^0), t \geqslant 0$ with a type-specific constant $T_k^0 \geqslant 0$, where θ denotes the Heaviside

Table 4.2: Date type characteristics and sensor presence on nodes.

Data Type	**Model**	**Rate**	**Wt.**	**Impact**		**Presence** [a]	
		r_k (B/s)	p_k	S_k^0 (m)	T_k^0 (min)	P_{mob}	P_{sta}
d_1–d_6	Analog	40	– [b]	500+	20	– [c]	1
d_7	Audio	107k	0.10	100	5	0.6	1
d_8	Picture	9k	0.14	10	60	0.9	0
d_9			0.12	750	30	0.3	0.7
d_{10}	Wi-Fi	900	0.06	60	20	1	1

[a] Represented by probability of presence on mobile and static nodes.
[b] Values of p_1–p_6 are respectively 0.16, 0.10, 0.10, 0.08, 0.08, and 0.06.
[c] Each mobile node has four out of six analog gas sensors on-board.

step function. We use the logarithm utility function $u(\boldsymbol{\omega}) = \log(1 + \sum \boldsymbol{\omega})$. Data type specs are shown in Table 4.2.

We acquired baselines from the naïve **everything** approach and a utility-driven **random greedy algorithm** that chooses sensors from cells where highest utility gain can be achieved until data quota is met, which degrades to "everything" with infinite quota. We compared them with HSF–ST and Lyapunov control algorithms. Test sets are designed to reveal the impact of (a) node mobility, (b) device heterogeneity, and (c) scale.

We also tested a basic genetic algorithm (GA) [102] with a limit on running time set to frame length T=1 min (Figure 4.3c). GA did not appear to fit in online scheduling and performed badly in most of the tests (e.g. Figure 4.3a). We believe this is because the time frame is too tight for GA to converge as the solution space expands exponentially when system scales up. Thus, GA is excluded from most of our results.

4.4.2 Performance Evaluation Metrics

Overall Performance

The **overall performance** is the optimization goal, given by the weighted sum of objectives as is formulated in Equation 4.6. Note that the overall performance considers both **benefit** and **cost**.

Coverage, Utility, and Number of Active Nodes

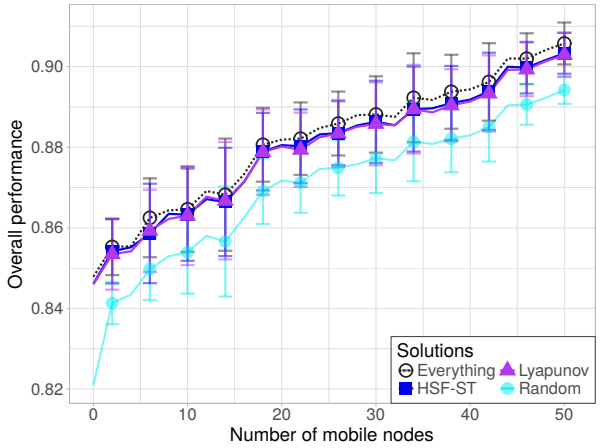
These are individual objectives defined in Section 4.2.2. We compare our approaches with them for in-depth analyses. Number of active nodes is normalized by $N \cdot K$.

Data Generation and Quota Utilization

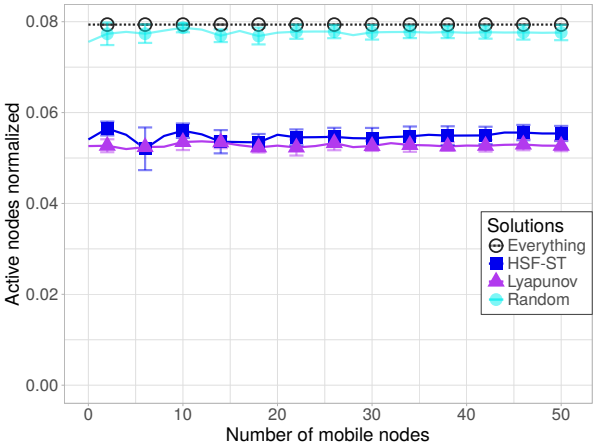
According to our formulation in Section 4.2.2, the operation is constrained by the data quota. Satisfying the long-term data quota constraint, a good algorithm usually maximizes its quota utilization.

Scheduling time

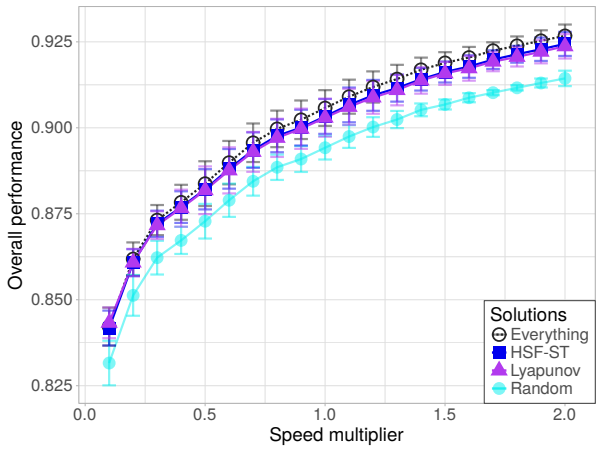
Scheduling time is the running time of the scheduling algorithm. In our proposed online approach, all computation needs to be done on the server within a limited time (shorter than the time frame). Hence, it is important to compare the running time of the algorithms.



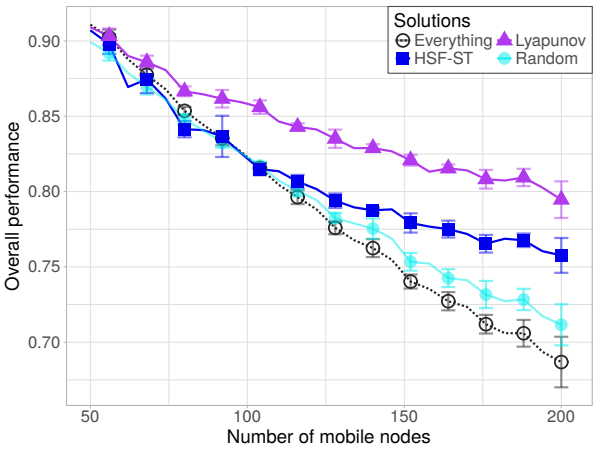
(a) Overall performance vs. node mobility, using fixed total number of nodes, N=50



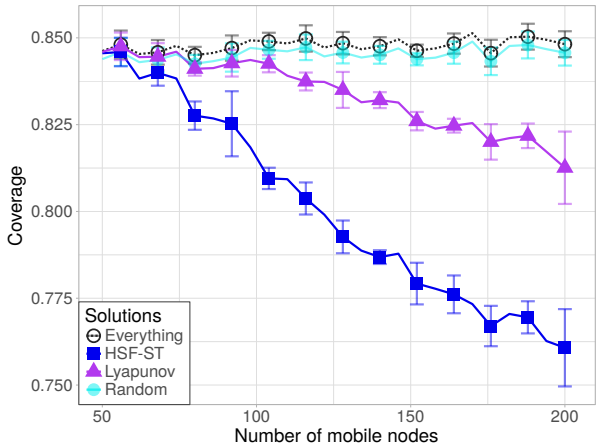
(b) Active nodes vs. node mobility, N=50



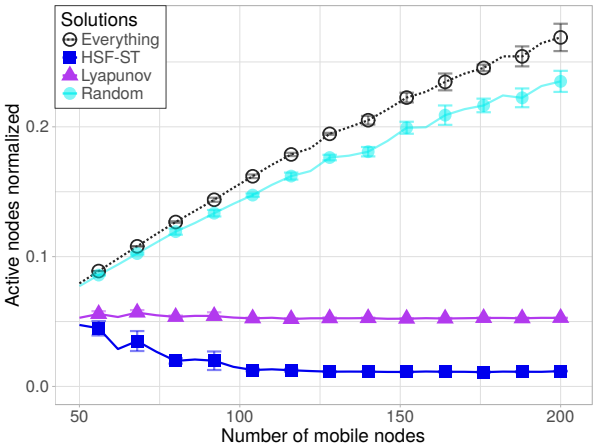
(c) Overall performance vs. speed of mobile nodes (multiplier$\times v \in [0.5, 1.5]$ m/s), N_{mob}=50



(d) Overall performance vs. device heterogeneity, using the same set of sensors



(e) Coverage vs. device heterogeneity



(f) Active nodes vs. device heterogeneity

Figure 4.2: Simulation results for impact of node mobility and device heterogeneity.

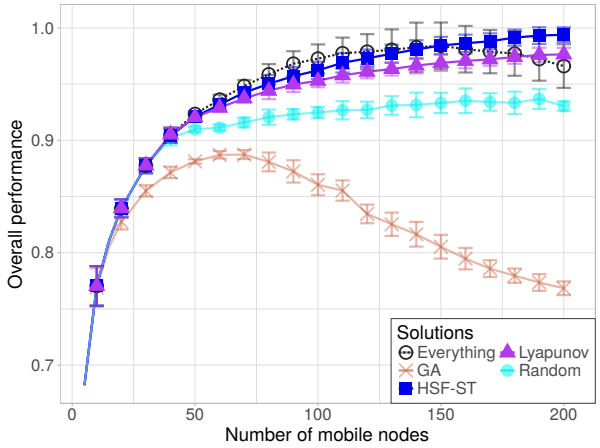
4.4.3 Simulation Results

Impact of Node Mobility

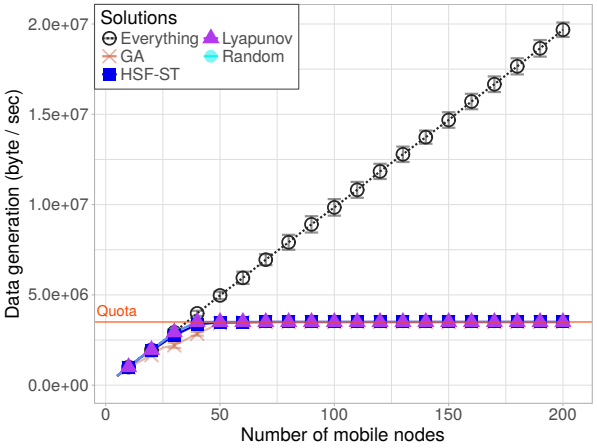
Figure 4.2(a,b,c) display the results on varying node mobility. In Figure 4.2a and 4.2b, the total number of nodes is fixed at N=50. As the number of mobile nodes among them increases, we notice that the overall performance increases for all tested algorithms (Figure 4.2a), while the average number of active nodes stays almost unchanged (Figure 4.2b). Similar trend has been observed when we increase the moving speed of the pedestrian model (Figure 4.2c, all nodes are mobile). These results show the benefits of using mobile sensing nodes as augmentation to existing in-situ deployments. In all these tests, HSF–ST and Lyapunov control show superior performance over the random greedy algorithm. Both our algorithms achieve overall performance close to "everything", while using 25–30% fewer active nodes.

Impact of Device Heterogeneity

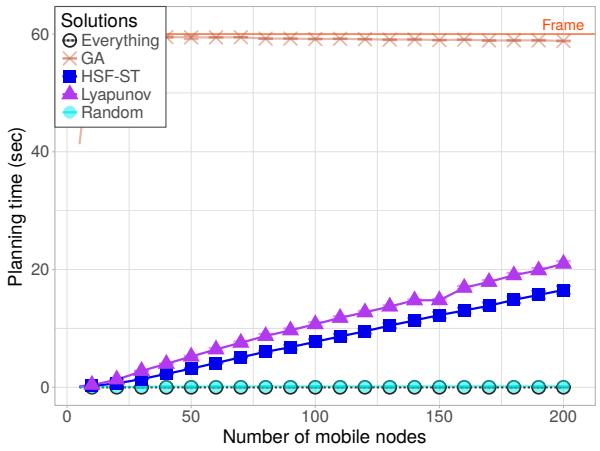
Figure 4.2(d,e,f) displays the results on varying device heterogeneity. Device heterogeneity is represented by the distribution of a set of sensors of each type across all nodes; all nodes are mobile in this test set. With more mobile nodes, the sensors are distributed more sparsely, so more nodes need to be activated (Figure 4.2f, especially the "everything" and "random" curve) to achieve the same level of coverage (Figure 4.2e). This means lower overall performance, because the overall performance considers the cost of node activation. In comparison, HSF–ST and Lyapunov control achieve better overall performance by balancing benefit (coverage and utility) and cost (node activation). The improvement is about 10% for N=200; this increases as the deployment scales up. Lyapunov control performs better than HSF–ST, especially when node heterogeneity increases, likely because the non-uniform distribution of sensors could trigger significant benefits that need to be balanced over time.



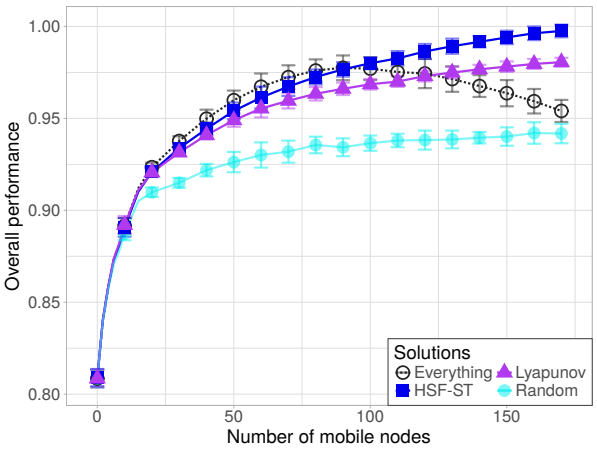
(a) Overall performance vs. total number of nodes



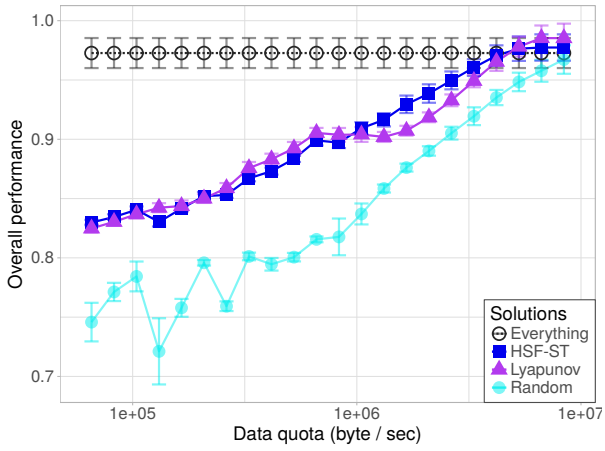
(b) Data generation vs. total number of nodes



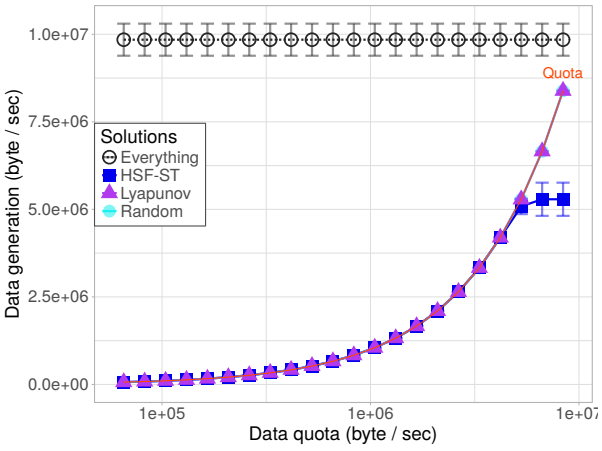
(c) Scheduling time vs. total number of nodes



(d) Overall performance vs. num. of mobile nodes



(e) Overall performance vs. data quota



(f) Data generation vs. data quota

Figure 4.3: Simulation results for impact of scale.

Impact of Scale

Figure 4.3 displays results for the impact of scale. Figure 4.3(a,b,c) are results for experiments where we tune the total number of nodes, with a fixed 40% of nodes being mobile. As the total number of nodes increases, the marginal performance becomes trivial and gradually gets overwhelmed by the penalty from node activation (Figure 4.3a, the "everything" curve). The random greedy algorithm does not scale, and its performance gets worse for $N \geqslant 40$. In comparison, HSF–ST and Lyapunov control achieve better-than-"everything" overall performance by nicely balancing benefits and cost. Figure 4.3b shows HSF–ST and Lyapunov control makes full use of the data quota without violation. Figure 4.3c shows a polynomial increase in scheduling times as predicted when the total number of nodes is small, and linear growth when the number of nodes approaches the "appropriate" value for the map. In this test set, HSF–ST seems to be slightly better than Lyapunov control, especially when the number of nodes is big. HSF–ST also runs about 25% faster than Lyapunov control in our settings. A trade-off between spatial and temporal resolution can be inferred here, i.e. a longer time frame should allow planning for more nodes on a large map with more cells.

Figure 4.3d is for a similar test set, but the number of in-situ nodes is fixed at 30 while the number of mobile nodes grows from 0 to 170. Figure 4.3e and 4.3f show performance and quota utilization when data quota increases from a very small value (hundreds of kilobytes) until it is more than sufficient. We find that Lyapunov control achieves better overall performance and quota utilization when data quota is larger. Both algorithms result in up-to 15% better overall performance as compared to the random greedy algorithm.

4.5 Summary and Discussion

In this chapter, we motivated and formalized the spatiotemporal scheduling problem in crowd augmented urban sensing systems. We proposed an online scheduling framework and two scheduling algorithms that address the challenges of heterogeneity and scalability. We leveraged the prototype platforms we designed and deployed in real community testbeds to collect measurements that were then used to drive extensive simulations. Experimental results showed that, in comparison to naïve approaches, the proposed algorithms (HSF–ST and Lyapunov) are significantly more efficient and scalable in heterogeneous community/city settings.

Future work aims at further addressing scalability issues through the use of scheduling hierarchies to offload work to edge servers and devices [44]. We also plan to explore the impact of uncertainties in network connectivity and node mobility on plan execution using different mobility prediction models. Such research is a key enabler to engaging human participation in smart community deployments.

Chapter 5

Data Upload Planning

As we discussed at the very beginning, IoT achieves its low cost and complexity by off-loading much of its logic to the cloud. Therefore, after data are generated/collected on end devices, they still need to be uploaded to the cloud (or published to the data exchange service in the SCALE architecture) for further analysis. In this chapter, under the community IoT context, we discuss how data upload can be appropriately planned to maximize the overall timeliness. Specifically, we address the optimized upload planning problem (i.e. determine optimal schedules for upload of gathered information to enable timely data collection in dynamic settings). We develop and validate a two-phase approach and associated policies, where an initial upload plan is generated with prior knowledge of upload opportunities and data needs, and a subsequent runtime adaptation phase alters the plan based on dynamic network and data conditions.

5.1 Chapter Overview

Enabling full-fledged deployment of IoT devices and supporting continuous operation of community scale IoT applications is difficult. We argue that effectively leveraging such mobile agents in smart communities [35] and planning their behavior with adequate prior knowledge can provide expanded coverage and reduced dependency on public infrastructures, and enable cost effectiveness in the collection of community related information. Alongside data collection, it is important that data get delivered to the backend (e.g. cloud services) in time to enable online analysis and ensure quick response.

Consider the case that people and mobile entities move around in the community – these could be buses or golf carts in everyday operation, or emergency vehicles and first responders in the case of an emergency event. We refer to them as mobile data collectors (MDCs). They carry heterogeneous devices with capability to collect data on their own or obtain data from deployed in-situ nodes. For simplicity, we assume that each MDC is provided with a trajectory that guides its travel. Data need to be collected at specific landmark points and uploaded to the backend using any available network in the community. The intermittent coverage of networks, unpredictability of data characteristics/needs, and dynamicity in the surrounding environment add to the challenges in determining how MDCs should operate and when and where to upload data.

This chapter focuses on techniques to ensure such timely and reliable data upload. The most straight-forward and naïve approach is to upload all data at a final collection point in the path (depot) or the first opportunity (completely opportunistically) and has been adopted in many existing works [27, 46]. However, such an approach fails to deal with the heterogeneous nature of networks, data, and applications, and fails to handle environmental dynamics effectively, resulting in inaccurate, incomplete or delayed information transfer. At the same time, mobility inherently causes the devices to be exposed to rapidly changing surroundings;

further more, moving patterns and trajectories of human data collectors can deviate from plans. These facts add to the heterogeneity and dynamicity of the system. Current efforts to design mobile data collection in large-scale events (e.g. earthquakes) or in communities with poor network coverage [134, 135, 67, 15, 48, 95, 123, 53, 126] focus on path planning and assume that the mobile data collectors (MDCs) have consistent connectivity to the backend server – this is unlikely to be true in real-world settings, especially in emergencies. Consistent and complete communication in all spaces is hard to create due to physical and policy limitations. Even in highly developed communities, public networks are not likely to be uniformly good.

In contrast, we optimally plan the upload of real-time information gathered by mobile data collectors that operate in conditions of intermittently available network contexts, and how to best leverage planning in realistic settings where dynamicity exists in network capacity (i.e. bandwidth), data characteristics (e.g. size, importance, timing), and movements. We present a two-phase approach to manage data upload for mobile data collectors to enhance the effectiveness of data collection in terms of timeliness, efficiency, and resilience. The key idea is to address the dynamically changing networks, data, and movements by exploiting prior knowledge of (a) community networks (e.g. location and quality of Wi-Fi access points or other upload opportunities), (b) the IoT deployments (e.g. where the sensors nodes are), and (c) the heterogeneous nature of IoT applications and associated data characteristics (e.g. volume and modality of data generated during a certain period). Prior knowledge could be learned from daily patterns of mobility and operation, and dynamics could be observed and resolved in real-time. Note also, that much community related IoT traffic (e.g. air quality data) is inherently delay-tolerant to some level. Leveraging any available information and context while being able to adapt to dynamics and uncertainties is the main focus of our proposed upload planning solutions.

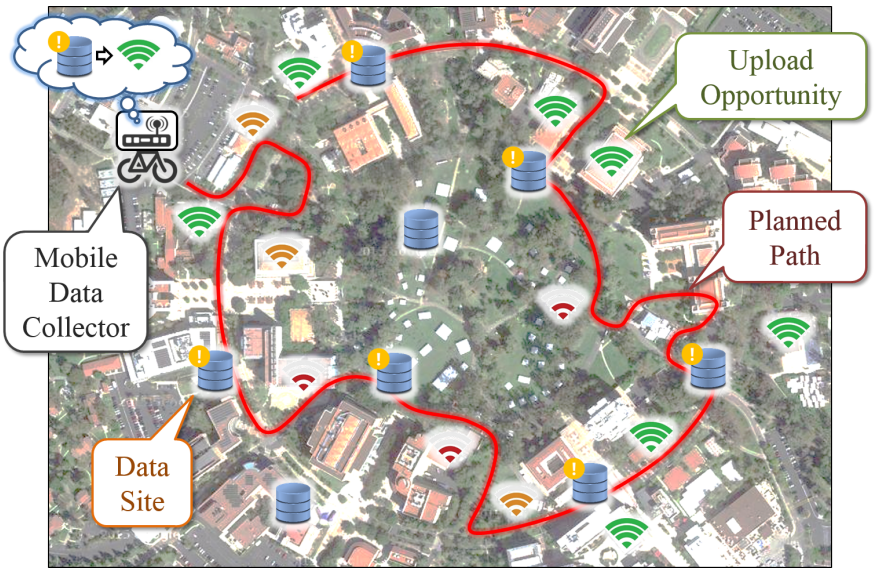


Figure 5.1: An upload planning problem, where data sites and upload opportunities lie on a planned path, and the MDC needs to decide where to deliver the data.

5.1.1 Problem Description in Detail

As illustrated in Figure 5.1, in our community data collection and upload scenario, the MDC travels through the community following a given path, where there are several places to collect data, and several "upload opportunities" that it can use to deliver the collected data to the backend. As we discussed previously, different types of data we would like to collect may have diverse degrees of importance and delay sensitivity. Upload opportunities enable Internet connectivity using multiple methods. These diverse approaches may lead to different costs in time, money, and energy. The community/city setting is complex, hence causes dynamicity in network connectivity and MDC mobility.

Knowledge about data (e.g. size, urgency) can be acquired from system specifications or application requirements. Deployed sensor nodes may also selectively send metadata to the backend based on bandwidth or cost limits. In general, we assume that knowledge about current data and the state of the system is available, albeit a little stale. Knowledge about opportunities can be estimated with system specifications, Internet service providers (ISPs), device reports through limited communication channels, or via crowdsourcing. The mobility

pattern of the MDC on a given path (i.e. the time it spends moving on each segment of the path), can be estimated with the knowledge about the MDC and the terrain, or past data from previous runs. Given such knowledge, we aim to plan the data uploading behavior of the MDC, to ensure timely delivery of critical data.

We formalize upload planning as a constrained optimization problem (shown to be NP-hard). The objective of planning is to determine the optimal schedule for data upload to maximize the overall data utility in terms of timeliness. While our work is motivated in a smart community scenario, this optimization problem can be applied to many mobile data collection and mobile sensing scenarios. For example, volunteers (mobile data collectors) can be dispatched in emergencies to take pictures or videos (data collection) at targeted locations, where public wireless networks may serve as "upload opportunities". Our scheme aims to assign several opportunities to each volunteer for data upload.

5.1.2 The Two-Phase Proactive Approach

Developing an optimal and comprehensive plan requires thorough knowledge about the state of the entire system, which is computationally complex and difficult to achieve in real-time. At the same time, due to the limited accuracy of predictions/estimations on future values of parameters (e.g. upstream data rate, data characteristics, moving time), there does not seem to be a straightforward way to determine whether a plan is optimal at any specific time. To handle the dynamic nature of the underlying networks, the application contexts, and the environment, we propose a two-phase proactive approach. In the first phase, referred to as **static planning**, a comprehensive plan is computed before the departure of an MDC at the server, based on collected information about the deployment and infrastructures and estimations of parameters. In the second phase, referred to as **dynamic adaptation**, the static plan is adjusted by the MDC at runtime. In this phase, the MDC executes the dynamic

adaptation schemes to adapt to the varying connectivity to the backend, determining how to best adapt the static plan based on data upload deadlines vs. the current timeline, and expectations on network availability and data characteristics vs. actual observations.

Key contributions of this chapter include:

- Formalization of upload planning (§5.2) as a constrained optimization problem that is NP-hard. A specific contribution here is that we have considered multiples types of runtime dynamics and addressed them in the formulation.
- Design of a two-phase solution and associated algorithms (§5.3). The solution combines a static planning stage that leverages known upload needs and opportunities and a dynamic adaptation phase that reconfigures the upload plan based on conditions at the upload site. A noteworthy contribution here is our Balanced Delay-Opportunity-Priority algorithm for static planning, with a Lyapunov-based control theoretic upload adaptation at runtime. The Lyapunov-based adaptation policy is general enough to handle common sources of dynamicity.
- Extensive performance evaluation of proposed techniques via simulations driven by real-world traces from deployments (§5.4) to demonstrate the effectiveness and investigate the stability and scalability of the combined solution in large community settings.

5.2 Upload Planning for Mobile Data Collection

In this section, we will discuss our assumptions to simplify the problem and define the terms and notations we used in the formulation. With the assumptions and notations, we formulate the upload planning problem as an optimization problem and prove it is NP-hard.

5.2.1 Assumptions

Exploiting the prior knowledge of upload planning requires comprehensive analysis that considers multiple factors. We make the following assumptions about our uploading planning problem to simplify the formalism.

We assume that: (a) We plan for **only one mobile agent** at any time. If there is more than one mobile agent, they work independently in isolation. (b) Tasks are short, so that power consumption and storage constraints are neglected. (c) A path planning phase exists a priori that generates a path for the mobile agent based on desired optimization needs. **This path is not changed** in upload planning. (d) Data are organized in **indivisible data chunks**. (e) Due to the limited range and obvious delay in connection setups, the mobile agent will have to **stop and stay** before the connection and data transmission take place. (f) During **static planning**, **we know necessary parameters** of data chunks, upload opportunities, and movements, which we refer to as their expectations or estimates.

5.2.2 Terms and Notations

A **mobile data collector (MDC)** is a mobile agent that is able to collect data and upload them to the backend server. It follows a **path**, which is an ordered sequence of geographical sites to visit. Since the mobile agent does not deviate from the path as we assumed, each point on the path can be mapped into a **location** x, which is the distance from the path origin to that point along the path. Note that the MDC may stop for communication at several points. When it is moving, its speed at any location x is $v(x), \forall x > 0, x \neq x(\boldsymbol{a}), \forall \boldsymbol{a}, x \neq x(\boldsymbol{u}), \forall \boldsymbol{u}$. We use the "speed vs. location" representation $v(x)$ here to indicate that the path and terrain are the dominant factors to limit the speed. In the static planning, $v(x)$ is estimated as $v_e(x)$.

A **data site** is a place where we are interested in sending the MDC for data collection. The data collection rates at all data sites are the same R_a. A **data chunk** consists of indivisible data that share some characteristics and have to be kept/transferred together. For simplicity, we assume each data chunk is held by a data site. A data chunk $\boldsymbol{a}$ can be described by quadruple $\boldsymbol{a} = (x, s, d, p)$, where x is the location of the data site holding $\boldsymbol{a}$, and s, d, and p are respectively the size, deadline, and importance level (priority) of $\boldsymbol{a}$. We assume $p \in (0, 1]$. The expectations of s and p are respectively s_e and p_e. Let $\{\boldsymbol{a}\}$ denote the set of all data chunks and N be the total number of data chunks, i.e. $N = |\{\boldsymbol{a}\}|$. We also use $x(\boldsymbol{a}_i)$ to refer to the x (i.e. location) of $\boldsymbol{a}_i$, and the similar notations for other attributes (size, deadline and priority). A data chunk is considered delivered when it is uploaded. The *actual* delivery time of $\boldsymbol{a}_i$ in a given upload plan $P = (\lambda, l)$ (defined later) is written as $t(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)$. The *estimated* delivery time used in the static planning is denoted as $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$.

An **upload opportunity** is a place where the MDC can upload data to the backend. An upload opportunity $\boldsymbol{u}$ is represented by $\boldsymbol{u} = (x, r)$, where x is its location, and r is its bandwidth (upstream data rate). Let $\{\boldsymbol{u}\}$ denote the set of all upload opportunities and M be the total number of upload opportunities, i.e. $M = |\{\boldsymbol{u}\}|$. We also use $x(\boldsymbol{u}_j)$ to denote the x of $\boldsymbol{u}_j$, and the same to r. The estimated uploading rate r_e is a very important piece of prior knowledge in the static planning. The data sites, upload opportunities, and speed function, that is, $\{\boldsymbol{a}\}$, $\{\boldsymbol{u}\}$, and v, together form the input of the upload planning problem.

An **upload plan** describes the upload activities of an MDC, which is ultimately the solution to a specific upload planning problem. A plan P is a tuple $P = (\lambda, l)$ where λ is the **global plan** that defines which data chunk goes to which opportunity, and l is the **local ordering** function that determines the order in which the chunks at the same opportunity are uploaded. Therefore, the global plan λ is a mapping over data chunks $\{\boldsymbol{a}\}$ to the set of upload opportunities, $\{\boldsymbol{u}\}$. We represent a global plan in three different ways for convenience

in different contexts. The first one is the mapping representation, where $\lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j$ if data chunk $\boldsymbol{a}_i$ is planned at opportunity $\boldsymbol{u}_j$. The second one is an N-by-M binary value plan matrix $\boldsymbol{\Lambda}$, where $\boldsymbol{\Lambda}_{i,j} = 1$ iff $\lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j$. The third representation is an N-dimensional plan vector $\mathbf{j} = [j_1, j_2, \cdots, j_N]^T$, s.t. $\boldsymbol{u}_{j_i} = \lambda(\boldsymbol{a}_i)$. The local ordering function $l(\boldsymbol{a}_i, \boldsymbol{a}_k, \lambda) = 1$ iff $\lambda(\boldsymbol{a}_i) = \lambda(\boldsymbol{a}_k)$, and $\boldsymbol{a}_i$ goes before $\boldsymbol{a}_k$ in the local order. To reduce the complexity of solutions, note that when uploading time is much shorter than moving time, l becomes less influential, so we can decouple it from λ. Therefore, in static planning, l is assumed to be a simple highest-priority comparator using deadline as a tiebreaker.

The constraint we have in the upload planning problem is the **cause-and-effect constraint**, which shows a data chunk cannot be uploaded before it is collected. The constraint can be represented as a binary matrix $\mathbf{C}$, $\mathbf{C}_{i,j} = 1$ iff $x(\boldsymbol{a}_i) \leqslant x(\boldsymbol{u}_j)$. A valid global plan λ, with its matrix representation being $\boldsymbol{\Lambda}$, may have $\lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j$, i.e. $\boldsymbol{\Lambda}_{i,j} = 1$, only if $\mathbf{C}_{i,j} = 1$.

5.2.3 Utility of Data Chunk Delivery

We argue that the utility of a data chunk depends on whether it is uploaded in a timely fashion or not. Therefore, we define a utility function, f, as a function of Δ, which is the difference between the delivery time and the deadline, i.e. $\Delta(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l) = t(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l) - d(\boldsymbol{a}_i)$. The utility function, f, should be chosen based on application-specific requirements. One important characteristic of the function is that it should be monotonically decreasing over $\Delta \in \mathbf{R}$, which indicates that utility declines as delivery gets late. For many applications, arriving early does not matter much as long as the chunk arrives prior to the deadline. In this way, $f(\Delta) = 1, \forall \Delta \leqslant 0$. In the planning phase, the utility can be estimated by $f(\Delta_e)$, based on the estimated delivery time, i.e. $\Delta_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) = t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) - d(\boldsymbol{a}_i)$.

We always hope to deliver all data chunks in a timely manner – when this is not possible,

we tend to find a plan that maximizes the overall utility of them.

The **weighted overall utility** (WOU) $U = U(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)$ of a plan λ is the weighted average utility of all data chunks, with weights being their importance levels, assuming local ordering l. i.e.

$$U = U(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l) = \sum_{i=1}^{N} p(\boldsymbol{a}_i) \cdot f\big(\Delta(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)\big) \Big/ \sum_{i=1}^{N} p(\boldsymbol{a}_i) \tag{5.1}$$

The WOU is used as a major performance indicator to compare various plans. A plan that ends up with a higher WOU is considered to be a better plan. Our objective is to find an upload plan for a MDC so as to maximize this WOU. However, when we make a plan before the MDC departs, due to the possible dynamics in the task which should not have occurred yet in this phase, the actual delivery time is unknown, so is the actual delay Δ. Therefore, in this phase, we have to use the **estimated weighted overall utility** (EWOU) $U_e = U_e(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ in planning heuristics as the objective function to maximize, which is

$$U_e = \sum_{i=1}^{N} p_e(\boldsymbol{a}_i) \cdot f\big(\Delta_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)\big) \Big/ \sum_{i=1}^{N} p_e(\boldsymbol{a}_i) \tag{5.2}$$

Unlike Δ, which is obtained directly at runtime,
$\Delta_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) = t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) - d(\boldsymbol{a}_i)$ has to be estimated by the planner. We provide detailed steps in §B to show how the planner makes this estimation.

5.2.4 Problem Formulation

With the assumptions and terms above, the upload planning problem is formulated as the following constrained optimization problem:

Given an ordered list of data chunks $\{\boldsymbol{a}_i\}, i = 1, \ldots, N$, with increasing $x(\boldsymbol{a}_i)$, and an ordered list of opportunities $\{\boldsymbol{u}_j\}, j = 1, \ldots, M$, with increasing $x(\boldsymbol{u}_j)$, we want to find plan λ and its corresponding plan matrix $\boldsymbol{\Lambda}$, to maximize the WOU $U(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)$ subject to the cause-and-effect constraint, i.e.

$$\max_{\lambda} \sum_{i=1}^{N} p(\boldsymbol{a}_i) \cdot f\big(\Delta(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)\big) \Big/ \sum_{i=1}^{N} p(\boldsymbol{a}_i) \tag{5.3}$$
$$\text{s.t. } \boldsymbol{\Lambda}_{i,j} \leqslant \mathbf{C}_{i,j}, \forall i = 1, \ldots, N, \forall j = 1, \ldots, M$$

The upload planning problem above is proven to be NP-hard, by showing the 0-1 knapsack problem can be reduced to an upload planning problem (§A).

5.3 Algorithms/Policies for Upload Planning

A naïve opportunistic approach is uploading *all* data chunks as long as there is an opportunity available. In this section, we will propose a family of feasible techniques, including two static planning algorithms and three dynamic adaptation policies, to address the problem using our two-phase approach described in §5.1.2.

5.3.1 Static Planning Algorithms

Static planning is the first phase of the proposed approach, which is executed offline on the backend server, before the departure of the MDC. Therefore, in this phase, it is acceptable and expected to optimize upload plans using comprehensive and computational intensive algorithms.

In this paper, we propose two computational techniques for constructing the upload plan.

One is based on the genetic algorithm (GA), and another uses an iterative greedy heuristic, Balanced Deadline-Opportunity-Priority (BDOP).

Genetic Algorithm

Due to the NP-hardness of our upload planning problem, we use heuristics to help with the optimization. Based on the definitions and problem formulation, we derived a simple implementation of a genetic algorithm (GA). The intuition here is, in a given problem setting, for each individual data chunk, some upload opportunities are generally "better" (though this is possibly affected by the schedule of other data chunks) than other opportunities, in terms of data rates and distance from the data site where the data chunk is collected. If we compare data chunks to loci (locations of genes) and different choices of opportunities to genes/alleles, then "better" opportunities are alleles who contribute more to the overall fitness. In this way, the upload planning problem is comparable to the evolution of a population, where each individual is a feasible solution to the optimization problem.

Based on such intuition, our genetic algorithm (GA) is derived as follows: Plan vector $\mathbf{j}$ is a *chromosome*, where j_i (the upload opportunity chosen for data chunk i) is a gene. An individual has only one chromosome. The EWOU $U_e(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ is used as the *fitness*. The initial population contains 100 individuals. Since it is easy to obtain a "plan vector" $\mathbf{j}$ that corresponds to the naïve approach – simply let $j_i = \min\{j | x(\boldsymbol{u}_j) \geqslant x(\boldsymbol{a}_i)\}$, we add such a "naïve plan vector" to the initial population to guarantee the bottom line of GA based optimization. In the reproduction phase, 5% elite individuals are kept as is, and 80% of children are generated from a random binary array based crossover with constraint checks.

According to §5.2.3, the time it takes to calculate the EWOU of a given plan λ is $O(MN^2)$. Therefore, the fitness calculation (bottleneck step) of each generation takes $O(MN^2)$ multiplied by the population size to complete. Note that it generally takes tens of generations

Algorithm 2 Balanced Deadline-Opportunity-Priority static planning algorithm for finding a plan λ to maximize $U_e(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$.

function planBalancedDOP ($\{\boldsymbol{a}\}$,$\{\boldsymbol{u}\}$)
Input : Data chunks $\{\boldsymbol{a}\}$ and upload opportunities $\{\boldsymbol{u}\}$
Output: Plan vector $\mathbf{j}$ corresponding to plan λ

```
Initialize N ← {a}.size, M ← {u}.size, and j ← [−1, ..., −1]^T_N
W ← {a}; // Put all chunks in {a} into W
while W is not empty do
    a_i ← Take the chunk with the earliest deadline in W
    Initialize S ← {}, totalSac ← 0, and b ← j
    loop
        u_j ← Fastest u to upload a_i in time, after other planned chunks
        if u_j is not null then
            b_i ← j; W ← S + W; j ← b; break; // Successfully planned a_i
        else
            a_k ← A planned chunk with least importance level to sacrifice
            if a_k is not null then
                totalSac ← totalSac + f(Δ_e(a_k, {a}, {u}, v_e, λ, l))
            if totalSec > p(a_i) then
                u_j ← Fastest u to upload a_i after other planned chunks
                b_i ← j; j ← b; break; // Too much sacrifice, give up a_i
            else
                S ← {a_k} + S; b_k ← −1; W ← {a_i} + W; // Sacrificed a_k
return j
```

of "evolution" before GA finds a maxima, so in practice, execution time of GA is usually longer.

Balanced Deadline-Opportunity-Priority

In this section we propose the **Balanced Deadline-Opportunity-Priority** (BDOP) algorithm. BDOP chooses data chunks in an earliest deadline first (EDF) based greedy manner, but it changes the plan when a previously planned low priority data chunk can be removed to fit in a high priority chunk, which we refer to as a *sacrifice*. BDOP is designed with the following concerns: (a) Data chunks with earlier deadlines should be uploaded at earlier opportunities, in case they expire in the future. (b) Upload opportunities with faster upstream

bandwidth should be preferred, in order to save time for other data chunks. (c) In limited time/resource, important data chunks should be given higher priority in planning compared to less important data chunks, in order to improve the overall utility of collected data.

This algorithm is sketched in Algorithm 2. BDOP starts with an empty plan $\mathbf{j}$, and all data chunks are put in set W. In every big loop (Ln 3–18), it takes out one $\boldsymbol{a}_i$ from W. First, the algorithm tries to find if $\boldsymbol{a}_i$ can be planned in time and after all other previously planned chunks. In fact, these chunks are not affected by $\boldsymbol{a}_i$ (Ln 7). If it succeeds, then $\boldsymbol{a}_i$ is planned. Otherwise, it tries to find a set of previously planned chunks to be sacrificed for $\boldsymbol{a}_i$ (Ln 10–18). At each iteration, a sacrificed chunk is picked in a lowest-priority-first manner, with the size being the tiebreaker (Ln 11). The algorithm then stays in the inner loop (Ln 6–18) until $\boldsymbol{a}_i$ can be planned in time (Ln 8), or the sacrifice is too big compared with the utility gain $\boldsymbol{a}_i$ brings (Ln 14). Then it either delays $\boldsymbol{a}_i$ and reverts sacrificed chunks (Ln 16), or plans $\boldsymbol{a}_i$ for in-time delivery and puts sacrificed chunks back into W (Ln 18). To simplify the computation of the utility gain brought by $\boldsymbol{a}_i$, we use $p(\boldsymbol{a}_i)$ as its estimation. Note that $p(\boldsymbol{a}_i)$ is the maximum possible utility gain.

In BDOP, $\boldsymbol{a}_k$ can be sacrificed for $\boldsymbol{a}_i$ only if it is less important than $\boldsymbol{a}_i$. Therefore, if $\boldsymbol{a}_k$ has been sacrificed for $\boldsymbol{a}_i$, $\boldsymbol{a}_i$ will never be sacrificed for $\boldsymbol{a}_k$. Note that sacrifice is monotonically made in an increasing order of importance levels, so $\boldsymbol{a}_k$ will not be sacrificed for $\boldsymbol{a}_i$ for more than once. Thus, the total number of sacrifices is no greater than $O(N^2)$. In each loop the time spent on calculating $f(\Delta_e)$ (the bottleneck step) is $O(MN)$. Therefore, the algorithm terminates in $O(MN^3)$ time.

5.3.2 Dynamic Adaptation Policies

Dynamic adaptation is the second phase in our two-phase approach and is executed on the MDC during task runtime. Hence, the policies for this phase should be simple enough to run

on mobile devices. Since the MDC cannot change the path or the points to visit, dynamic adaptation only happens at upload opportunities. In this phase, more information becomes available in addition to the prior knowledge: (a) The MDC has a static plan λ. (b) When the MDC arrives at an opportunity $\boldsymbol{u}_j$, it has information regarding the *actual* characteristic of data that will be uploaded. (c) It is able to estimate the bandwidth r as it uploads. This phase is intended to keep the benefits of the static plan (e.g. the global view of the task in optimization) while adapting to possible dynamics in networks, data, and movement along the task execution.

In this part we design three adaptation policies, namely (a) strict to the plan, (b) strict to the plan's timeline, and (c) an adaptively balanced policy based on Lyapunov control.

Strict Static Plan

Strict static plan represents a purely planned approach. It does not adapt to runtime dynamics, but makes full use of the static plan. In this policy, when the MDC arrives at $\boldsymbol{u}_j$, it tries to upload all $\boldsymbol{a}_i$ s.t. $\lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j$ as specified by plan λ. If $\boldsymbol{u}_j$ is not available for any reason, these data chunks will not be uploaded in later opportunities, but carried back by the MDC physically.

Strict Timeline

The strict timeline policy is another attempt to stick to the original plan. Instead of keeping the plan unchanged, the policy maintains the plan's timeline. In other words, we try to complete all uploads at each opportunity $\boldsymbol{u}$ within the expected time given by plan λ. To implement this, we need to compute the expected completion time, $t_e(\boldsymbol{u}_j, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda)$ as

follows.

$$t_e(\boldsymbol{u}_j, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda) = \max\{t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) | \lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j\} \tag{5.4}$$

In the actual implementation, $t_e(\boldsymbol{u}_j)$ for each opportunity can be calculated by the static planner in the planning phase, so it would not become an extra overhead for the MDC.

If the MDC has finished all data chunks planned at the current opportunity but there is still time left, it tries to find some high-priority data chunks (better still within their deadlines) to upload with the objective of achieving higher utility.

Adaptation Using Lyapunov Control Strategy

We devise another dynamic adaptation technique using Lyapunov control [68]. Lyapunov control is usually applied to systems that evolve over time with active "queues". The overall goal of the framework is to maximize the time-averaged reward under the constraint that all "queues" remain bounded. These queues are modeled as system states, and their growth refers to the system's tendency towards instability. The framework defines a Lyapunov function over the system states (i.e., the current queue sizes) and tries to keep the Lyapunov drift, the expected difference between the Lyapunov function values at two successive steps, as small as possible, which ultimately ensures the system reaching its goal over time.

In our upload plan, we define the state of our system, $\phi(t)$ at time slot t, by a vector of two queues: $\phi(t) = [Q(t), T(t)]$. A time slot t actually corresponds to an upload opportunity $\boldsymbol{u}$ encountered by our MDC. Queue $Q(t)$ refers to queue backlog at the MDC, i.e. the total size of items yet to be uploaded, and $T(t)$ measures the amount of time elapsed since the MDC started its operation. Let β be the time that is supposed to have elapsed according to the static plan λ constructed a priori. This value is updated at every upload opportunity

when the MDC arrives there. Ideally, we want the MDC to take actions to keep $Q(t)$ low and $T(t)$ close to β. Hence, a quadratic Lyapunov function is defined as

$$L\big(\phi(t)\big) = \frac{1}{2}Q^2(t) + \frac{1}{2}\big(T(t) - \beta\big)^2 \tag{5.5}$$

Then the Lyapunov drift is

$$\Delta(t) = \mathbf{E}\big[L\big(\phi(t+1)\big) - L\big(\phi(t)\big)\big|\phi(t)\big] \tag{5.6}$$

Let $R(t)$ define some reward function that the system tries to maximize. Then, the strategy to adopt according to the Lyapunov theory [68] is to take actions at time slot t that minimizes

$$\Delta(t) - V \cdot R(t) \tag{5.7}$$

with some control parameter, V. At each upload opportunity, our MDC chooses items to upload, which reduces $Q(t)$ but increases $T(t)$. Let $X(t)$ be the total size of the items selected to upload. Therefore, we have $Q(t+1) = Q(t) - X(t)$, $T(t+1) = T(t) + X(t)/r(t)$, where $r(t) = r(u)$ is the data uploading rate. It is shown that $\Delta(t) \sim B + \mathbf{E}\big[-Q(t)\cdot X(t) + \big(T(t) - \beta\big)\cdot X(t)/r(t)\big]$, where B is a constant that approximates $\big(1 + 1/r^2(t)\big)\cdot \mathbf{E}\big[X^2(t)\big]$. In our case, the reward $R(t)$ is the total utility of selected data chunks. Let $\sigma(\boldsymbol{a})$ be 1 if chunk $\boldsymbol{a}$ is selected (and 0, otherwise), then we obtain

$$X(t) = \sum_{\boldsymbol{a}} \sigma(\boldsymbol{a}) \cdot s(\boldsymbol{a})$$
$$R(t) = \sum_{\boldsymbol{a}} \sigma(\boldsymbol{a}) \cdot p(\boldsymbol{a}) \cdot f\big(T(t) + s(\boldsymbol{a})/r(t) - d(\boldsymbol{a})\big)$$

Hence minimizing Equation (5.7) is equivalent to maximizing

$$\sum_{\boldsymbol{a}} \left(\sigma(\boldsymbol{a}) \cdot s(\boldsymbol{a}) \cdot \left(Q(t) - \frac{T(t) - \beta}{r(t)} \right) + V \cdot p(\boldsymbol{a}) \cdot f\left(T(t) + \frac{s(\boldsymbol{a})}{r(t)} - d(\boldsymbol{a}) \right) \right) \tag{5.8}$$

which reaches its maxima when we let $\sigma(\boldsymbol{a})$ be 1 for each $\boldsymbol{a}$ that makes the summed expression in Equation (5.8) positive.

In actual implementation, each time the MDC arrives at an opportunity or finishes uploading a data chunk, it checks its buffer and picks up the data chunk that results in the maximum positive value for the summed expression in Equation (5.8) and loops until no such chunk is found. This takes $O(1)$ for each data chunk since all the parameters should be known to the MDC when this adaptation occurs, and takes at most $O(N)$ at each upload opportunity, for all data chunks collected but yet to be uploaded. Since $Q(t)$, $T(t) - \beta$, and $p(a) \cdot f$ all have different scales and units, we need to normalize them with some coefficients. The tuning results are, $V = 1$, the coefficient of $Q(t)$ be set to the magnitude of 10^{-6}, and that of $T(t)$ be set to the magnitude of 10^{-4}. The performance is not very sensitive to coefficient changes as long as they are at those magnitudes.

5.4 Performance Evaluation

Due to the limited scale of real-world deployments, the relatively long time to run real experiments, and the lack of diversity in real settings, we derive synthetic configurations for simulated experiments to demonstrate the effectiveness of our proposed approach. Simulation parameters have been designed/tuned to track the real-world observations and measurements to best reflect the real-world scenarios and environment.

5.4.1 Experimental Environment and Simulation Setup

Our simulations are mostly done in the QualNet simulator [83]. Data exchange is implemented in a customized application-layer traffic generation protocol in the user library. All wireless communication goes through 802.11a/g. Bandwidth and transmission powers are tuned to reflect the data uploading rate and RSSI we measured in real settings. The QualNet simulations were done on a Dell OptiPlex 755 PC, which has an Intel Core 2 Duo E6550 dual core 2.67 GHz CPU, and 4.00 GB of DDR2 SDRAM. The OS is Ubuntu 14.04 LTS 64-bit. The simulator is a QualNet EDU 7.3 compiled from the source with our customized user application.

Since QualNet EDU license allows only up to 50 nodes per scenario, and the simulation takes long time, we could not scale up the experiments in QualNet. Thus, medium and large cases were simulated in MATLAB R2015B. However, comparing results on small test sets from QualNet simulations and from MATLAB simulations, we observed significant correlation – the results were mostly similar in values and consistent in trends. Therefore, we believe the MATLAB simulations provide reasonable estimations on actual performance in medium and large settings.

Using data collected on community testbeds, we created test cases at multiple scales as is shown in Table 5.1. We compared our two-phase approach with the naïve and purely opportunistic approach (referred to as "first opportunity"). For our two-phase approach, since we have two algorithms for static planning and three policies for dynamic adaptation, there will be six combinations to compare: GA only, BDOP only, GA-ST, BDOP-ST, GA-Lyapunov, and BDOP-Lyapunov.

To test the effect of network availability and capacity, the different approaches are compared in experiments with different number of upload opportunities (3–30 for small cases, 4–40 for medium cases, and 10–200 for large cases) and different standard deviation of uploading

Table 5.1: Specifications of small, medium, and large test sets.

Scenario Characteristics		**Small**	**Medium**	**Large**
MDC	Constant Speed V/ms^{-1}	5 [a]		
	Est. Conn. Time T_c/s	12		
Path	Length Exp. $\mathbf{E}[L]/km$	12	24	200
Data Sites and Chunks	Bandwidth $R_a/Mbps$	12		
	Total Number N	120	120	1,000
	Distance $\Delta x(\boldsymbol{a})/m$	$N(90, 20^2)$	$N(180, 60^2)$	
	Size s/MB	$U(2, 8)$ [a]		
	Importance Level p	$\{0.3, 0.6, 1.0\}$ [b]		
	Deadline Inc. $\Delta d/s$	$U(-40, 200)$		
Opportunities	Total Number M	3–30 [a]	4–40 [a]	10–200 [a]
	Distance $\Delta x(\boldsymbol{u})/m$	$U(0, 2\mathbf{E}[L]/M)$ [a]		
	Bandwidth $r_e/Mbps$	$N(4, 2^2)$ [c]		

[a] Subject to change if this parameter is chosen as an independent variable.
[b] Importance level is chosen from this set with probabilities 0.6, 0.3, 0.1, respectively.
[c] The minimum valid bandwidth is 200 Kbps, and the same to other test cases.

bandwidth (0–2.23 Mbps for all cases). To test the effect of data heterogeneity and data dynamics, the approaches are compared using different average size of data chunks (1–10 MB for all cases) and different level of dynamics in data size. To test the effect of mobility dynamics, we experiment with different number of upload opportunities (4–40 for medium cases) and different level of dynamics in movement. To test the scalability of the approaches, we use different number of upload opportunities (the same settings for medium and large cases) and different number of data chunks (10–200 for medium cases and 100–2,000 for large cases).

5.4.2 Performance Evaluation Metrics

We use three metrics to compare the performance of studied approaches: the weighted overall utility (WOU), the fraction of important data chunks uploaded in time, and the total time spent to complete all data collection and upload.

Weighted Overall Utility (WOU)

In our experiments, we used the utility function f defined below as a piecewise function, which is flat for $\Delta \leqslant 0$ and decreases exponentially for $\Delta > 0$.

$$f = \begin{cases} 1 & , \Delta \leqslant 0 \\ \exp(-\Delta/T_f) & , \Delta > 0 \end{cases} \tag{5.9}$$

where $1/T_f > 0$ is the attenuation coefficient. In our tests, we used $T_f = 30/\ln 2$, so the utility of a single data chunk halves every 30 seconds after its deadline.

The weighted overall utility (WOU) is a comprehensive performance index we defined in Equation 5.1 to measure the overall performance of a plan. In dynamic simulations, WOU can be calculated from statistics. In static simulations, we calculate EWOU $U_e(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ from Equation (5.2) for comparison.

Fraction of Important Data Chunks Uploaded In Time

The fraction of important data chunks that are uploaded in time is a straightforward metric. Intuitively, an intelligent plan should accommodate important chunks first to maximize the overall utility. This fraction can be calculated by comparing $t(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)$ with $d(\boldsymbol{a}_i)$. In static tests, $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)$ is used in place of t.

Time Consumption to Complete All Data Collection

The total time required to complete all data collection can also be used as a benchmark to evaluate the planning algorithms, as we expect the MDC to complete the assigned task in as short time as possible. The total time is the completion time at the last opportunity, $t(\boldsymbol{u}_M, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda)$. In MATLAB static simulations, its estimation $t_e(\boldsymbol{u}_M, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda)$ is used.

5.4.3 Simulation Results

With the simulation results we have, we are able to compare static planning algorithms presented in Section 5.3.1 and dynamic adaptation policies discussed in Section 5.3.2 respectively to validate and demonstrate the applicability and effectiveness of our proposed algorithms in both phases. We also compare our two-phase approach using different algorithm combinations, with corresponding static-only (completely planned) approaches and the naïve (completely opportunistic) approach to show the effectiveness and competitive performance of the two-phase optimization.

As we proceeded with our experiments, algorithms or algorithm combinations that often appeared to be defective (e.g. worse than the naïve approach) or obviously inefficient were removed from future comparisons, which allowed us to focus on techniques that worked well.

Basic Comparisons and Effect of Network Availability

In these basic comparisons, we first compared algorithms for the two phases separately, and then compared our two-phase approaches with static-only approaches and the naïve approach (a.k.a. "first opportunity"). These results are displayed in Figure 5.2 and 5.3.

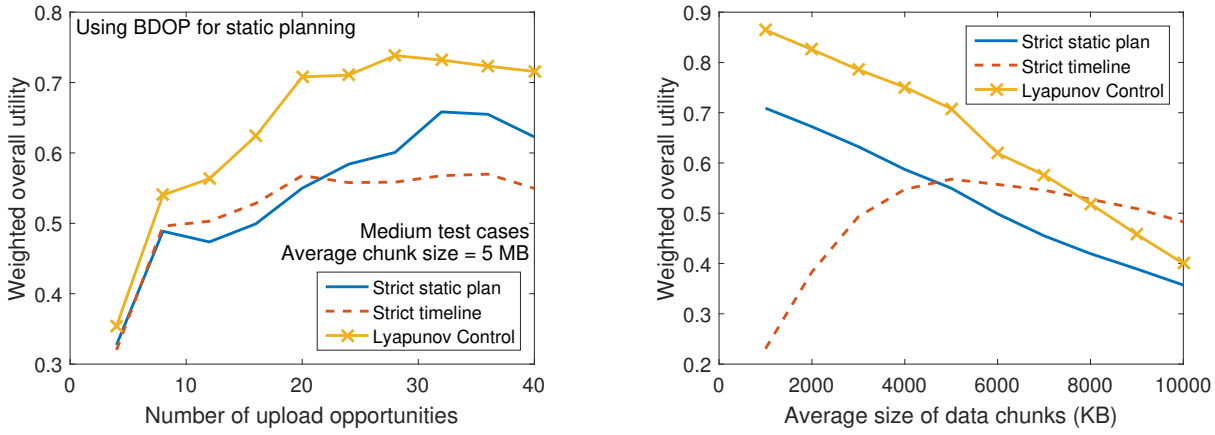


Figure 5.2: Simulation results for effect of network availability and capacity, using BDOP for static planning, comparing WOU of dynamic adaptation policies.

Figure 5.2 shows the results on medium sets that compares the three dynamic policies. In all cases, the static plans were generated by the Balanced Delay-Opportunity-Priority (BDOP) algorithm. Results show that the Lyapunov optimization performed better than the "strict static plan", and that the "strict timeline" policy exhibited the worst performance, especially for larger number of upload opportunities as shown in Figure 5.2a and for smaller data chunks as shown in Figure 5.2b. Since the "strict static plan" represents the completely planned approach and requires least computation, we concluded that "strict timeline" was not effective, and would be omitted from future experimentation. Comparisons of static algorithms on small sets show that GA and BDOP performed similarly well; hence other combinations with both these static planning algorithms were compared with the naïve approach.

In Figure 5.3, we compare the performance of multiple combinations of static planning algorithms and dynamic adaptation policies, as the number of upload opportunities increases. Figure 5.3a shows the QualNet simulation results on a small test set. GA-Lyapunov and BDOP-Lyapunov performed equally well. Compared with the naïve approach, the improvement in WOU was about 14–24%. Also note that both of the two-phase approaches

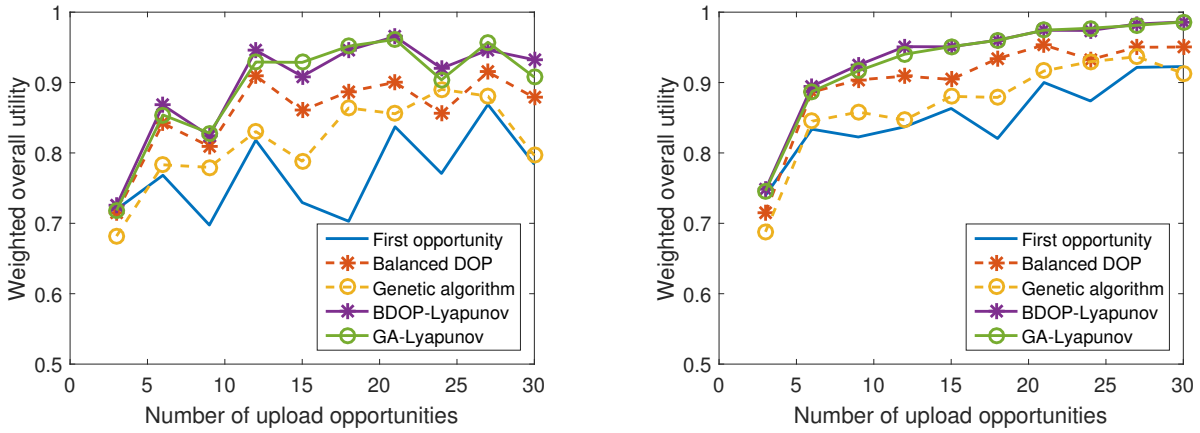


Figure 5.3: Simulation results for effect of network availability and capacity, comparing WOU of multiple approaches.

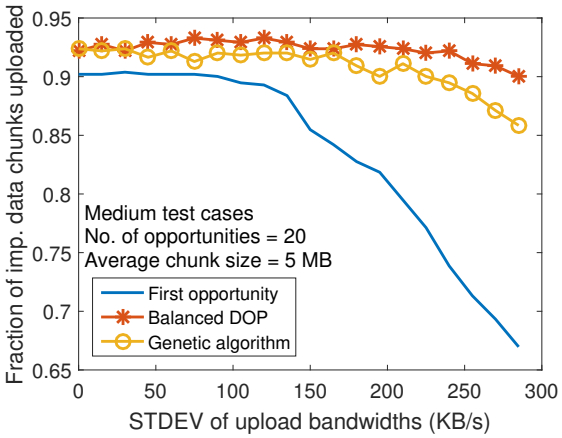


Figure 5.4: Simulation results for effect of variance in network availability.

performed better than corresponding static-only approaches, which shows the effect of the dynamic adaptation phase in our two-phase approach. Figure 5.3b shows the MATLAB simulation results on the same test set. Compared to the QualNet experiments, MATLAB simulations modeled fewer details of the physical networks and protocol stack implementation. This led to less fluctuation and relatively better results. We also note that the relative positions and trends of the lines were similar.

Figure 5.4 shows the effect of network heterogeneity and variance in network availability. Two static algorithms were compared with the naïve approach in a medium test set. We can

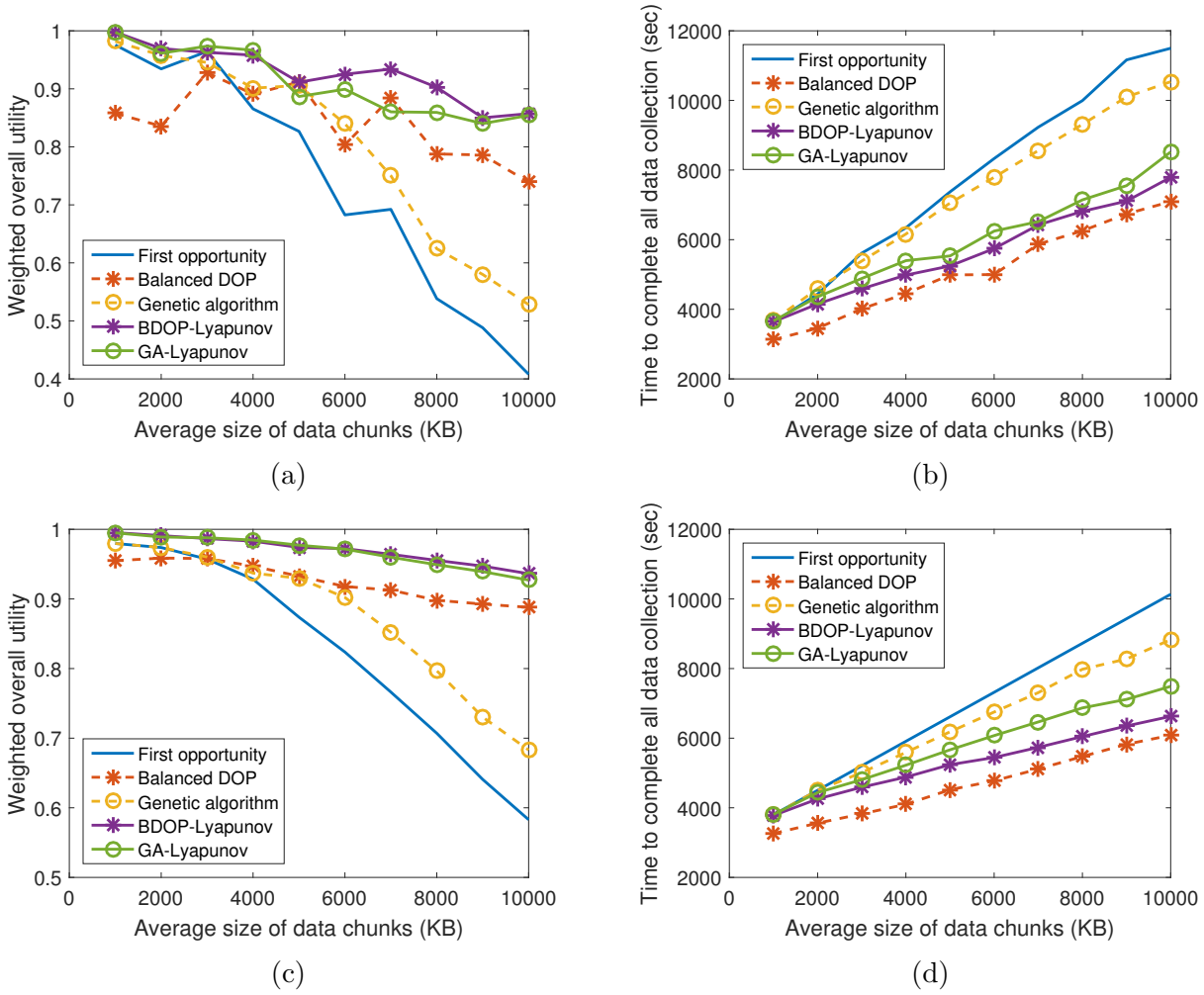


Figure 5.5: Simulation results for effect of data heterogeneity.

see both BDOP and GA performed much better than the naïve approach, especially when the variance among upload opportunities increased. The reason might be that planning with a global view of all data chunks and upload opportunities (by the static planner) enables the MDC to leverage the faster opportunities and avoid the slower ones, while there was no way for the opportunistic naïve approach to do the same. This comparison shows the significance of a global plan, especially in communities and cities where connectivity is not uniformly good.

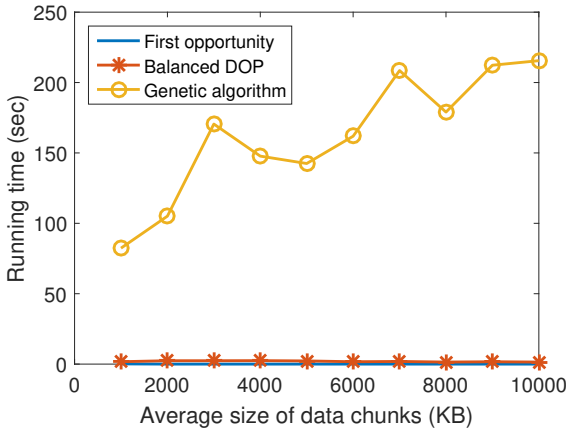
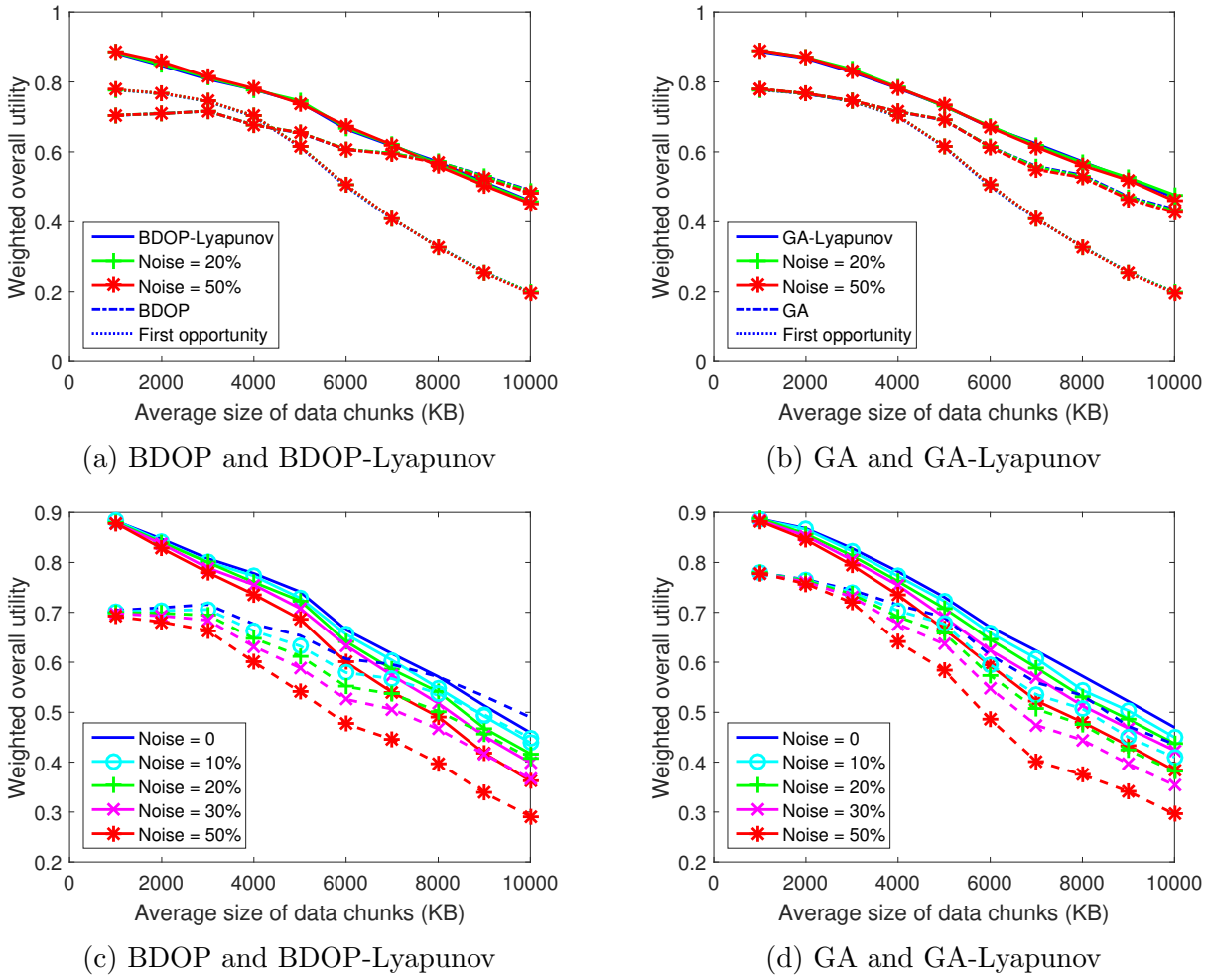


Figure 5.6: Comparison of running time of static planning algorithms.

Effect of Data Heterogeneity

Results in Figures 5.5a and 5.5b show the QualNet simulation results for our two-phase approach were more stable as the size of chunks increased, compared with the static-only approaches and the naïve approach. The improvement in WOU can be as big as 36–60% for larger data chunks, and the reduction in total time was up to 30%, compared with the naïve approach. Interestingly, though the BDOP based approaches (static and two-phase) performed similarly well with GA based approaches in terms of WOU as is shown in Figure 5.5a, BDOP seemed to save a little more time than GA in Figure 5.5b. Figures 5.5c and 5.5d are MATLAB simulation results on the same test sets. Again, we can see the relative positions and trends of the lines were mostly the same. Together with results in Figure 5.3b, the use of MATLAB simulations on larger-scale experiments (e.g. in Section 5.4.3) is justified.

Figure 5.6 compares the running time of static planning algorithms, where GA ran much slower than BDOP. Results also show that, as the complexity of problem increased (e.g. data chunks became larger), the running time of GA increased like linearly. With larger test sets, we found it took more than 20 minutes to generate a plan for one scenario with 1,000 data chunks using GA, but the performance outcome was not better than BDOP (actually



(a) BDOP and BDOP-Lyapunov

(b) GA and GA-Lyapunov

(c) BDOP and BDOP-Lyapunov

(d) GA and GA-Lyapunov

Figure 5.7: Simulation results for effect of different levels of data dynamics. CS stands for "chunk size".

slightly worse, perhaps due to the increased size of the solution space). Therefore, we will not evaluate GA in scalability studies.

Effect of Data Dynamics

Figure 5.7 displays two sets of simulation results on a medium test set, showing the effect of dynamics in data chunks (i.e uncertainties in chunk size and data priority). Different markers represent different levels of dynamics. The dynamics are implemented by the half proportional range of the uniformly distributed noise added to the size of each data chunk, this

also represents the probability that the priority of each data chunk is elevated by one level. Different line styles represent different approaches: solid lines for **two-phase approaches**, dashed lines for **static-only approaches**, and dotted lines for **the naïve approach**.

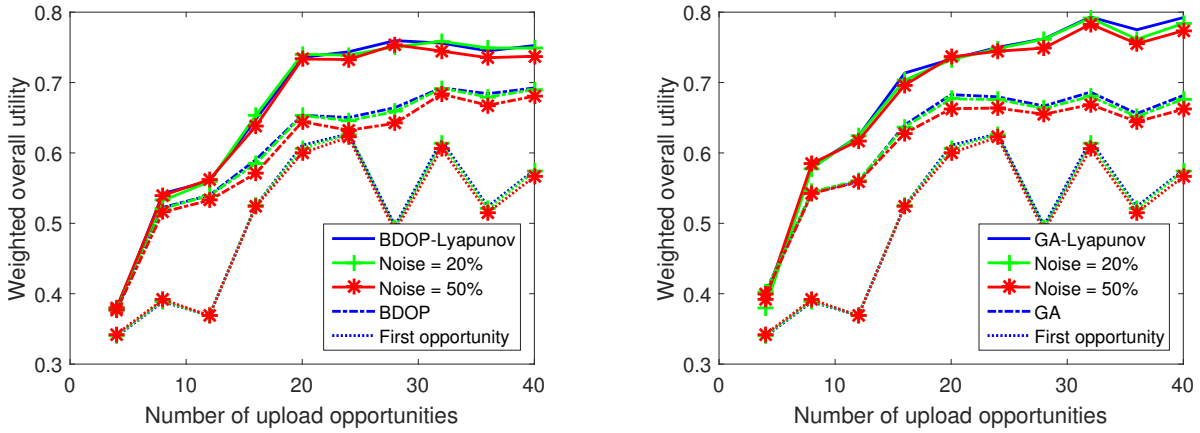
In the first set of experiments (Figures 5.7a and 5.7b), the expectation of chunk size remained unchanged. In the second set of experiments (Figures 5.7c and 5.7d), the expectation increased by the half proportional range. For example, if half range $l/2 = 20\%$ chunk size, a data chunk that was expected to be 5 MB may end up being $[5 \times (1 \pm 20\%)]$ MB in the first experiment, or $[5 \times (1 + 20\% \pm 20\%)]$ MB in the second experiment, and a data chunk of medium importance may turn out to be of high importance with a chance of 20%.

In Figures 5.7a and 5.7b, the three groups of tightly bundled curves in both figures suggested the dynamics in chunk size and data priority had limited effect on the overall utility of collected data, as long as the noise added to chunk size had a mean value of 0.

Figures 5.7c and 5.7d show the two-phase approaches using Lyapunov for dynamic adaptation always result in higher WOU, compared to corresponding static-only approaches, when chunks are bigger than expected in average. The decrease in WOU caused by data dynamics is also smaller in tests using two-phase approaches, which means having Lyapunov control based dynamic adaptation for the second phase improves the resilience of planned operation over data dynamics.

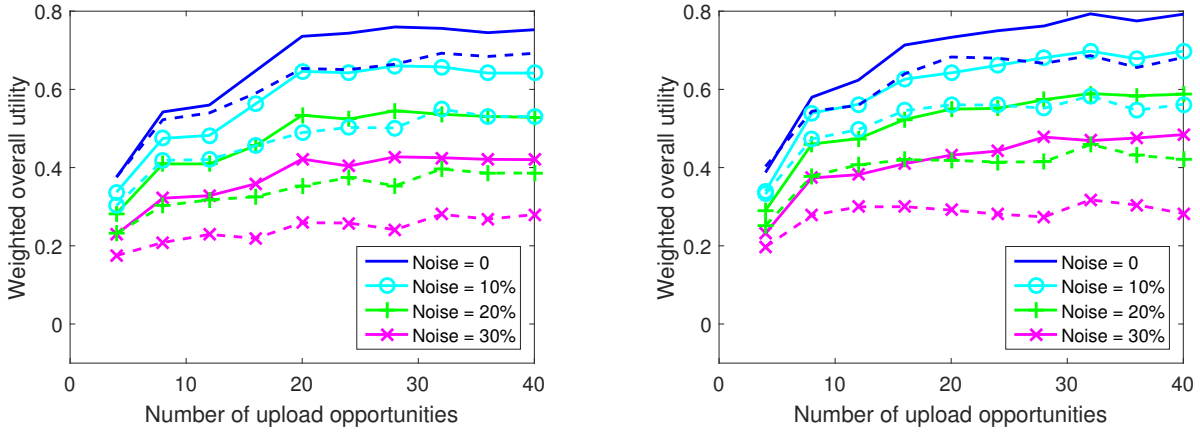
Effect of Mobility Dynamics

Figure 5.8 displays two sets of simulation results on a medium test set, showing the effect of mobility dynamics (i.e uncertainties in moving time). We illustrate results under different levels of dynamics (i.e. noise). In the graphs, different line styles represent different approaches: solid lines for **two-phase approaches**, dashed lines for **static-only approaches**, and dotted lines for **the naïve approach**.



(a) BDOP and BDOP-Lyapunov

(b) GA and GA-Lyapunov



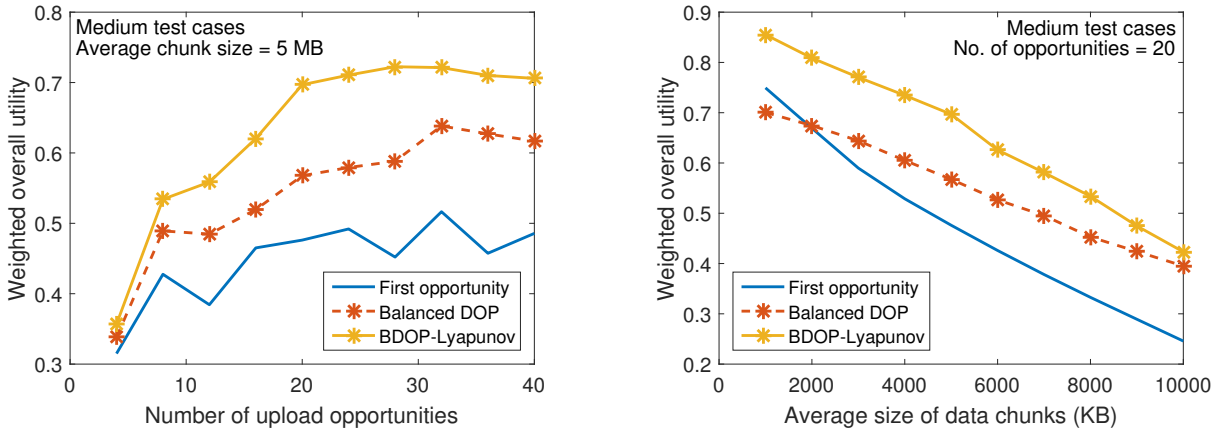
(c) BDOP and BDOP-Lyapunov

(d) GA and GA-Lyapunov

Figure 5.8: Simulation results for effect of different levels of dynamics in moving time.

In the first set of experiments (Figures 5.8a and 5.8b), the expectation of moving time was unchanged. In comparison, in the second set of experiments (Figures 5.8c and 5.8d), the expectation of moving time increased. For example, under standard deviation $\mu = 20\%$ moving time, a movement that was expected to take 100 sec may end up taking $[100 \times (1 - 20\%) + y]$ sec in the first experiment, or $(100 + y)$ sec in the second experiment, where y is an exponentially distributed random variable with $\lambda = 1/\mu = 1/20$.

In Figures 5.8a and 5.8b, the three groups of tightly bundled lines in both figures suggested the dynamics in moving time had limited effect on the overall utility of collected data, as long as the noise had a mean value of 0.



(a) (b)

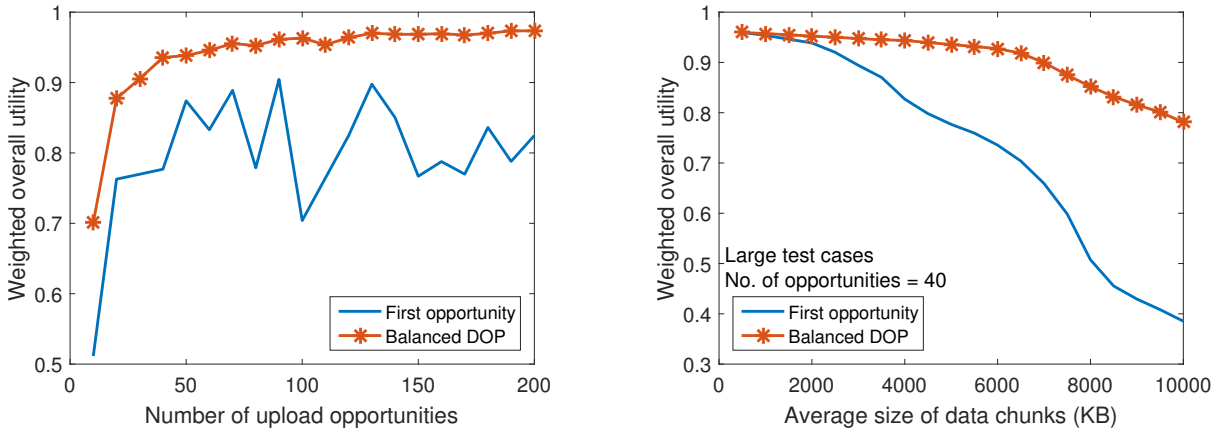
Figure 5.9: Simulation results for demonstration of scalability.

Figures 5.8c and 5.8d (when moving time is longer than expected in average) show that the two-phase approaches which use the Lyapunov control for dynamic adaptation always resulted in higher WOU, compared to the corresponding static-only approaches. Similar to the results for data dynamics, the decrease in WOU caused by mobility dynamics is also smaller with the two-phase approaches, which means using Lyapunov control based dynamic adaptation for the second phase also improved the resilience of planned operation under mobility dynamics.

Scalability

Results in Figures 5.9 and 5.10 show that our approach worked well as the scale of deployment increased. Figure 5.9 shows the results for medium test sets, and Figure 5.10 for large test sets. In Figure 5.9a, the WOU was improved by 38–46% by our two-phase approach when there were more than eight opportunities, in comparison to the naïve opportunistic operation. In comparing Figure 5.9 with Figure 5.3 allows us to conclude that the advantage of our two-phase approach was more obvious when the deployment scales up.

Figure 5.10 compares the performance of BDOP with "first opportunity" on large test sets,



(a) (b)

Figure 5.10: Simulation results for demonstration of scalability on large test sets.

and shows that BDOP performs well and is more stable as the system scales up. In summary, our two-phase approach using BDOP-Lyapunov exhibits superior performance as compared to other strategies.

5.5 Summary and Discussion

In community deployment settings, one can exploit planned mobility (e.g using regularity of transit vehicles and community volunteers) for data gathering and upload planning; in such settings, there is a priori knowledge of communication network availability and mobility paths/trajectories. Our findings through measurement studies also provide practical guidance on how this mobility should be tailored. We stress on the realistic "stop–operate–go" mobility pattern to ensure complete data collection. The two-phase upload planning approach and associated scheduling algorithms and adaptation policies are based on factors derived from real world measurement studies. In our work, we also highlight the need for timely data delivery under non-uniform network settings and show how planned mobility can be utilized to realize the dynamic communication needs. In this paper, we hone in on a specific problem – the upload planning problem in smart community IoT deployments and

propose a two-phase approach to solve the optimization in community/city settings. We design SCALECycle, a prototype system, to conduct measurements in our real community testbeds. Experiments using data collected with SCALECycle, show that our two-phase approach using BDOP-Lyapunov outperforms both the naïve approach and the planned approach, in complex community settings of different scales with multiple different types of dynamics.

Our future work will leverage the use of multiple MDCs and multiple access networks in a cooperative manner. Currently, we assume that the path planning phase is done separately a priori, potentially by other planners; hence, we decouple the two aspects of path and upload planning and focus on details and realistic issues associated with upload scheduling. If multiple MDCs are all given different paths and work independently with the fixed paths, the proposed techniques and individual scheduling approaches will work. An interesting direction of future work is the collaboration of multiple MDCs that enables joint data collection and upload. Multiple MDCs become handy when the trajectories can be manipulated or MDCs could be arranged to deviate from their original plans to capture dynamic events – this brings about multiple questions. Which MDCs should have their path/plan modified? Should this be done for each MDC one at a time or should we design a joint planning mechanism? Should path planning and upload scheduling for each MDC be conducted in separate phases or jointly? How can MDCs be made to deviate from their planned and expected paths as new data collection events arise? In fact, we have addressed similar path planning and detour planning problems in previous efforts [137] without considering data uploading. In future work, we plan to study the interaction between MDCs (e.g. their collaboration and competition) and its impact on upload planning and path planning. Integrating a growing number of mobile devices into a changing and heterogeneous IoT ecosystem requires innovative networking approaches as well. New network architectures and protocols to handle wireless access networks into IoT settings are being designed and developed [118, 97, 58]. Our recent work on multi-network IoT deployments [90] illustrates

the role of upcoming technologies such as SDN in managing the heterogeneous nature of IoT systems. Our future directions also include incorporating a multi-network upload setting to improve the overall efficiency of data collection in large-scale community deployments, which could be the key to driving future smart communities worldwide.

Chapter 6

Sensor Calibration Planning

Now we have planning techniques to improve the efficiency and timeliness of data delivery. However, data lose their utility if the accuracy and spatiotemporal consistency are not guaranteed. In this chapter, we discuss how to make efficient sensor calibration (i.e. maintenance) plans in the long-term operation of community-scale IoT systems. Specifically, we design and implement an efficient cooperative approach to solve the calibration planning problem, which aims at minimizing the cost of the recurring calibration of multiple sensor types in the long-term operation. We design and validate a two-phase solution that consists of a sensor selection phase that minimizes the average cost of recurring calibration, and a path planning phase that minimizes the travel cost of multiple calibrators which have load constraints.

6.1 Chapter Overview

The advent of IoT ecosystems with low cost sensors and actuators is enabling the widespread deployment of IoT-enabled smart spaces in our homes, communities and cities. In this

chapter, we address cost-accuracy issues that arise in the deployment of affordable IoT systems. In particular, the aggregation of relevant knowledge at scale from low cost smart spaces is problematic since low cost sensing solutions imply low accuracy, faster degradation and drift. The relative inaccuracy of the connected devices can be alleviated through the automated and in-situ calibration of the sensors – made possible due to the linearity of the bias in most cases [106]. For example, mobile platforms (including humans that carry calibration devices) can visit the smart spaces, either opportunistically or in a planned manner, to assess the biases of the deployed sensors and compensate for errors [69]. One can thus generate "sufficiently accurate" knowledge over time by frequent calibration of the low cost sensors in the deployed infrastructure; however, the low deployment cost might result in increased maintenance costs! Careful planning of the calibration process is therefore essential for cost-effective monitoring of the smart spaces – this is increasingly important as the number and size of smart spaces grow.

In this chapter, we address the calibration planning problem. The aim is to develop a plan for the calibration of a large number of inexpensive (and often inaccurate) sensors in a smart space using high-integrity reference sensors that are mobile, such that (a) the deployment and operational costs for calibration are minimized while (b) maintaining a sufficient observation accuracy from the sensor measurements. More realistically, given knowledge of a sensor's degradation characteristics, we program its calibration with respect to an observed phenomenon so as to maintain adequate sensing accuracy while minimizing the required effort from the mobile calibrators. We formalize the above as a multi-path planning problem which we solve via intelligent heuristics, and validate using real world settings. Our key contributions include:

- A measurement study to help understand the calibration issue of low cost sensors for environmental monitoring, which motivates the proposed approach for scalable calibration of IoT-enabled smart spaces (§6.2).

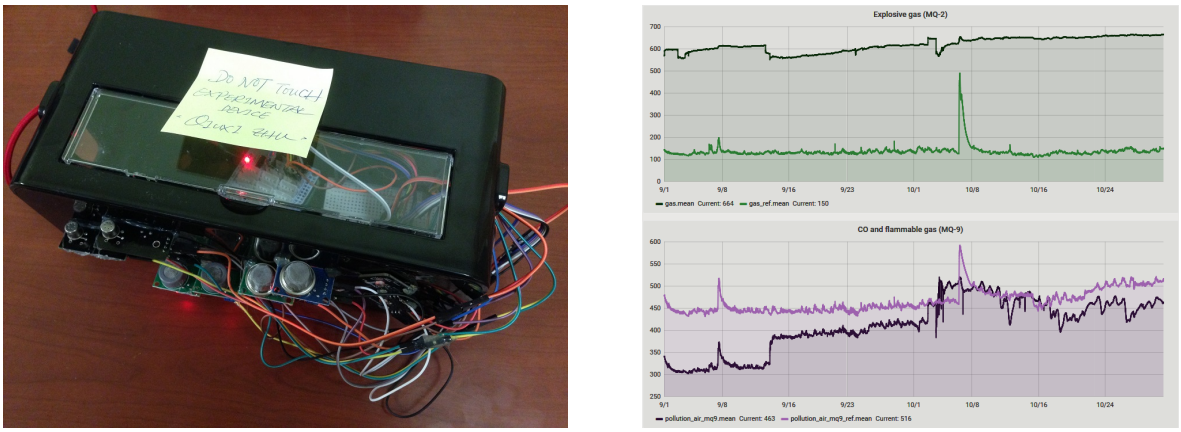
- Long-term multi-sensor calibration planning as a service that exploits device locality, sensor characteristics, and application needs.
- The characterization and formulation of the multi-sensor calibration planning problem (§6.3) and discussion of NP-hardness.
- A two-phase iterative solution and a family of heuristic methods to enable the cost-effective planning of multi-sensor calibration in large smart spaces and over longer time periods (§6.4).
- The validation of our approach and algorithms leveraging real-world indoor and outdoor smart spaces settings from our ongoing testbeds in Irvine, CA and Paris, France; results show significant cost improvement compared to other approaches while maintaining adequate accuracy.
- Initial steps towards a prototype of a calibration planning service for IoT smart spaces that features: (i) a dashboard used to map the deployment of sensor nodes over large spaces and plan the supporting multi-sensor calibration, and (ii) a mobile app that the calibrators (mobile workers) use along their calibration journeys (§6.5).

6.2 Background Experience

Experiences in crowdsensing and IoT deployments indicate that, in general, measurements gathered from low-cost sensors are inaccurate and deviate from the ground truth. Our first experience in calibration came from lessons learned while deploying IoT-enabled environmental sensing in the SCALE project [11, 115], which has been discussed earlier in §3.3. The SCALE deployments allow us to obtain rich information about the sensor behaviors and especially their respective deterioration across time in various environments. In all cases, we observed a deterioration in the response of the low-cost gas sensors over time.

Our second experience in calibration comes from the launch of an urban-scale experiment with the city of Paris in 2015 [86]. As part of our efforts, we developed a crowdsensing application for mobile phones for monitoring the exposure of the urban population to environmental noise pollution [37]. While crowdsensing may be a cost-effective approach for monitoring urban environmental conditions (e.g., noise) that exhibit high spatio-temporal variability, the relatively low quality of the embedded sensors as well as the uncertain sensing context required dedicated actions. We quickly learned (as anticipated) that only a low percentage of the crowdsensed measurements actually contribute to the analysis of the urban noise pollution. We needed to enhance the quality of the gathered observations [47]. As a first step, we thoroughly studied the bias in smartphone noise sensing against a reference sound level meter, which helped us develop a protocol for calibrating individual handsets [121]. Through experimental studies, we demonstrated that a calibration protocol can help gather observations that can more accurately map noise pollution at the district level [119]. Unfortunately, the proposed calibration protocol places a high demand on the end-users who must actively participate in the calibration. We subsequently investigated a distributed protocol for opportunistic multi-party calibration of devices located in the same sensing and communication range [99].

To validate our observations and generate a custom model of sensing deterioration, we developed a measurement platform to reproduce the deterioration process. Here, the custom sensor box was instrumented with 4 pairs of low-cost sensors as a "test node" (Figure 6.1a); data was obtained from this node for 6 months. Figure 6.1b shows the data reported by two pairs of MQ gas sensors 3–4 months after the installment: long-running MQ-2 sensors that were connected to a running node for more than two months generated different values at the same location and show significant different in sensitivity when exposed to similar stimuli. In summary, our prior experience with IoT-enabled smart spaces (indoor and outdoor) indicates that low-cost sensing may bring valuable information; but with time, obtaining accurate results required analysis of the contributed measurements and error compensation.



(a) (b)

Figure 6.1: UCI testbed in Donald Bren Hall (Dept. of Computer Science), showing (a) a node deployed in our lab, where we reproduced the sensor deterioration process; (b) readings from two pairs of low-cost sensors on the test node, 3 months after their installment.

In contrast to the related efforts on sensor calibration (§2.3), we take a more holistic approach to low-cost calibration in smart spaces at scale. First, we consider the presence of heterogeneous sensor types with varying calibration characteristics. Our proposed approach is application-aware and is able to take into account diverse sensing needs (sensor type, sensing accuracy) presented by the context at hand. We assume the ability to utilize multiple mobile calibrators. We exploit locality of in-situ sensors (that can calibrate each other) to reduce the cost of mobile calibrators. The multi-step approach presented next highlights the overall scheme for a more comprehensive and efficient calibration planning mechanism for IoT-enabled smart spaces.

6.3 Multi-Sensor Calibration Planning

Our objective is to enable the cost-efficient calibration planning of a given smart space with multiple types of sensors and diverse applications. The target environment assumes the in-situ deployment of a large number of inexpensive sensors with an adequate number of high-precision *mobile calibrators* (e.g., trained workers that calibrate the in-situ devices).

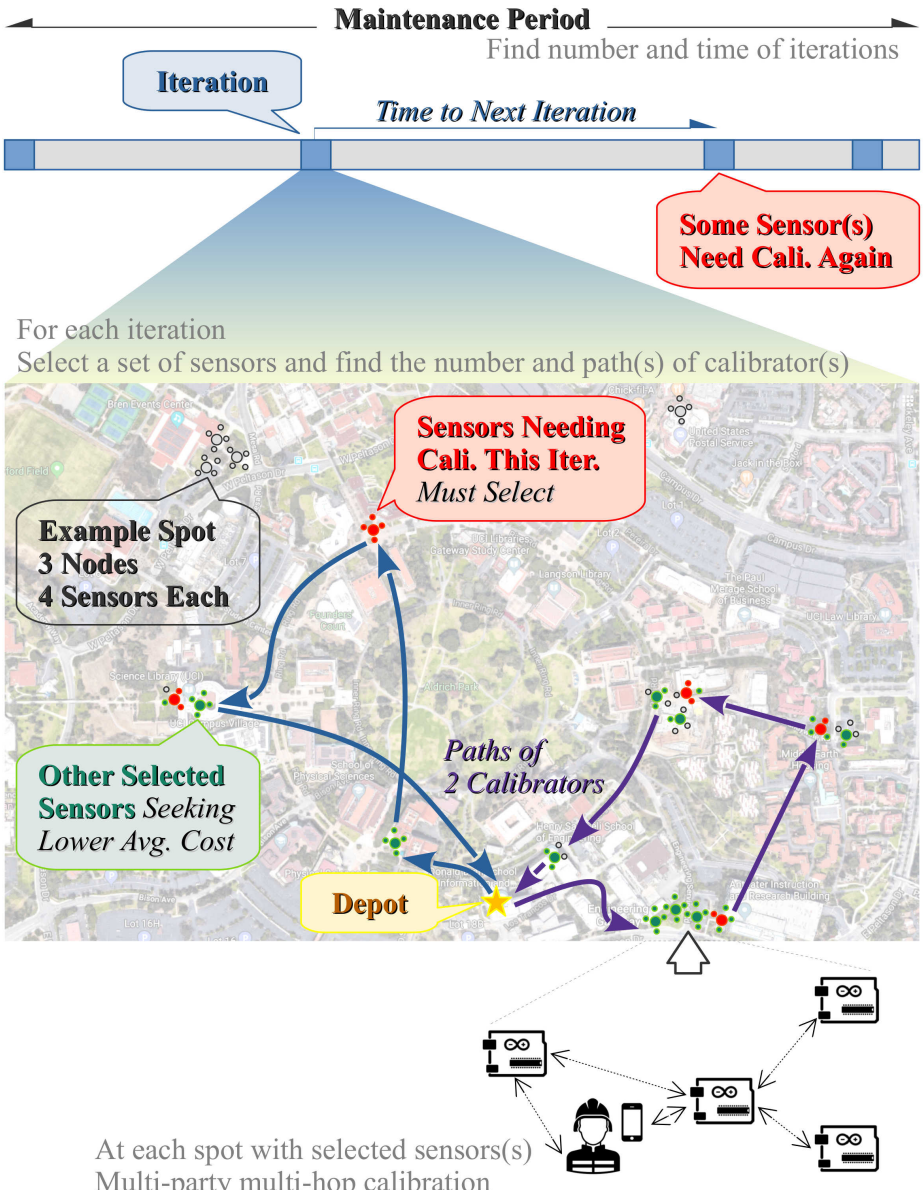


Figure 6.2: Calibration planning over a long maintenance period.

We propose an effective approach to calibration planning over a long term duration, i.e. a given **maintenance period** T: (i) First, we partition the space to form calibration *spots* by exploiting locality of in-situ sensors; a sensor can calibrate other sensors of the same type at the same spot. (ii) Next, we carry out multiple *iterations* of calibration during the given maintenance period T. Our challenge lies in determining the number of iterations and the time at which each iteration is executed. Furthermore, in each iteration we determine – the sensors to calibrate, the number of mobile calibrators needed, and the paths taken by each mobile calibrator. The sensor calibration is then carried out accordingly over a number of Ω **iterations** so that the overall cost over T is kept to a minimum and ensures that all the sensors always comply with the *data accuracy requirements*. The planning problem then amounts to identifying the least number of iterations and planning the calibration at each iteration ω=$1, 2, \ldots, \Omega$ so that the number of workers involved is also kept to a minimum. A sketch of our approach is shown in Figure 6.2.

Next, we introduce the notations and assumptions underlying our formalization of the multi-sensor calibration planning problem.

6.3.1 Notations and Assumptions

A **node** n_j, j=$1, 2, \ldots, N$, is an IoT device that embeds one or more types of low-cost **sensors**. A **sensor type** s_k, k=$1, 2, \ldots, K$, refers to the capability of detecting a certain type of phenomenon (e.g., temperature, gas concentration), which usually requires a specific kind (or combination) of low-cost sensor(s). Then, we denote an individual sensor by $n_{j,k}$, and we introduce the binary **sensor presence** matrix $\mathbf{Q}_{N \times K}$ to characterize the set of available sensors so that $Q_{j,k}$=1 (resp. 0) if sensor type s_k is present (resp. absent) on node n_j.

Calibration-Related Terms

A **reference sensor** is a "standard" high-quality sensor, whose readings serve as the ground truth and therefore can be used to calibrate other sensors that sense the same phenomenon. We assume that the reference sensors are calibrated offline (e.g., in a lab before each iteration) and their calibration is out of the scope of our work.

A **mobile calibrator** (or simply a *calibrator*) m_i is a person who carries reference sensors and visits the field to calibrate the deployed low-cost sensors. For simplicity, we assume that a calibrator can be equipped with any of the reference sensors needed across his/her journey. **Sensor calibration** takes place when a calibrator m_i visits a node n_j and stays at the spot for long enough to calibrate a sensor $n_{j,k}$.

Each sensor type s_k is associated with a **calibration time** τ_k that ranges from a few seconds to several minutes, depending on the phenomenon detected by the type of sensor and the calibration complexity. We associate each sensor type s_k with a **calibration period** T_k, which characterizes the maximum duration for which the sensors remain valid (i.e. the measurements have sufficient accuracy) once calibrated. The definition of the period depends on the usage scenario and may be learned from empirical study.

During operation, each individual sensor $n_{j,k}$ is associated with a **time to next calibration** (TTNC) $F_{j,k}$ that indicates how soon the sensor needs to be (re-)calibrated. The TTNC matrix $\mathbf{F}_{N\times K}$ then represents the TTNCs of all the sensors. $\mathbf{F}$ is a function of time, where each $F_{j,k}$ decreases between iterations and is reset to T_k when $n_{j,k}$ is calibrated. We further denote $\mathbf{F}[\omega_-]$ (resp. $\mathbf{F}[\omega_+]$), the matrix TTNC immediately before (resp. after) the iteration ω. Note that $\mathbf{F}$ is non-negative, i.e. $F_{j,k}[\omega_\pm]\geqslant 0$, $\forall(j,k)$, $\omega\in\mathbf{N}_+$. If node n_j does not hold a sensor of type s_k, the corresponding TTNC is infinite, i.e. $F_{j,k}[\omega_\pm]=+\infty$, $\forall\omega\in\mathbf{N}_+$, if $Q_{j,k}=0$.

A **sensor selection** is a collection of sensors (selected for calibration) represented by a

binary matrix $\mathbf{\Gamma}_{N\times K}[\omega]$, where $\Gamma_{j,k}\leqslant Q_{j,k}$. If sensor $n_{j,k}$ is selected for calibration at iteration ω (i.e. $\Gamma_{j,k}[\omega]=1$), then, at the end of this iteration, its TTNC is reset to its calibration period T_k, i.e. $F_{j,k}[\omega_+]=T_k$; otherwise, its TTNC stays unchanged (during iteration ω) because we assume the duration of an iteration is much shorter than the gap between iterations:

$$\mathbf{F}[\omega_+] = \mathbf{\Gamma}[\omega] \circ \mathbf{T}_N + (1 - \mathbf{\Gamma}[\omega]) \circ \mathbf{F}[\omega_-] \tag{6.1}$$

Where $\mathbf{T}_N$ is the nodal calibration period matrix that consists of N identical rows of $[T_1, T_2, \cdots, T_K]$; "$\circ$" is the element-wise multiplication of matrices. Typically, there is no need to run an iteration if no sensor needs immediate calibration; also, special needs and unexpected changes could be easily addressed by altering the TTNC matrix. Hence, the **time to next iteration** (TTNI) after ω is the minimum TTNC of all the sensors, i.e. $t_{\omega+1}-t_\omega=\min \mathbf{F}[\omega_+]$; thus, the TTNC matrix immediately before the next iteration is $\mathbf{F}[(\omega+1)_-]=\mathbf{F}[\omega_+]-\min \mathbf{F}[\omega_+]$.

Smart-Space-Related Terms

We abstract a smart space as a set of spots that are such that sensor nodes deployed at a **spot** ν_l, $l=1, 2, \ldots, L$, are sufficiently co-located to enable their concurrent calibration by a single calibrator. Note that this may possibly involve leveraging multi-party multi-hop calibration (§6.2). We then map a smart space as directed graph $G=(V, E)$, where each vertex $\nu_l \in V$ corresponds to a spot and each edge weight (ν_{l_1}, ν_{l_2}) denotes the average time taken by a calibrator to move from ν_{l_1} to ν_{l_2}. The graph can be represented as an adjacency matrix $\mathbf{G}_{L\times L}$, where G_{l_1,l_2} is the weight on the directed edge (ν_{l_1}, ν_{l_2}) (i.e. the movement cost). Note that the physical movement from ν_{l_1} to ν_{l_2} may pass through other spots if that is the fastest option. Therefore, as long as each spot is physically accessible, G is a complete digraph and all its edges have finite weights.

The node **locations** are represented by a binary location matrix $\mathbf{D}_{N\times L}$, where each row represents a node and each column a spot. We set $D_{j,l}$=1 if node n_j is deployed at spot ν_l; or 0 otherwise. A node is deployed at a single spot, so $\sum_{l=1}^{L} D_{j,l}$=1. For simplicity, we assume that all the calibrators depart from the same spot, which is referred to as the "depot" in related work on path planning, and is conventionally indexed as the first spot, i.e. ν_1. We assume that no node is deployed at the depot. Given the node locations $\mathbf{D}$, we can derive the **spot selection** vector $\mathbf{h}_L$ from the sensor selection $\mathbf{\Gamma}$. A spot is selected for iteration ω if any sensor on any node deployed at that spot is selected in $\mathbf{\Gamma}[\omega]$ i.e.:

$$h_l[\omega] = \bigvee_{j=1}^{N} \bigvee_{k=1}^{K} \Gamma_{j,k}[\omega] \cdot D_{j,l} \tag{6.2}$$

In each iteration, the **path** of a calibrator is an ordered sequence that starts from the depot and visits a set of non-repeating selected spots. It can be represented as a binary matrix $\mathbf{W}_{L\times L}$, where W_{l_1,l_2}=1 if the calibrator visits spot ν_{l_2} immediately after visiting ν_{l_1}; or 0 otherwise. We require that all mobile calibrators return to depot in the end, i.e. $\sum_{l=1}^{L} W_{1,l} = \sum_{l=1}^{L} W_{l,1}$. The path of calibrator m_i in iteration ω is denoted $\mathbf{W}_i[\omega]$. Each selected spot is visited exactly once by one calibrator:

$$\sum_{i=1}^{M} \sum_{l=1}^{L} W_{l,l_0,i}[\omega] = h_{l_0}, \;\; l_0 = 2, 3, \ldots, L \tag{6.3}$$

The other constraints that the matrices $\{\mathbf{W}\}$ should satisfy are discussed in §6.4.2.

6.3.2 Definition of the Calibration Cost

We consider three major types of cost: *iteration overhead*, *movement cost*, and *calibration cost*. The **iteration overhead** C_{it} is the cost of all the activities related to an iteration that are not tied to any specific calibrator, such as preparation, equipment, and the transport to

the deployment, etc.

The **movement cost** C_w corresponds to the cost associated with the travel time of the calibrators while moving between spots. The **movement time** $C_{w,i}[\omega]$ of a single calibrator m_i in iteration ω can be computed from the map G and the calibrator's path $\mathbf{W}_i[\omega]$. It equals the sum of the weights on the edges between all the consecutive pairs of spots visited by the calibrator:

$$C_{w,i}[\omega] = \sum_{l_1=1}^{L} \sum_{l_2=1}^{L} \left(W_{l_1,l_2,i}[\omega] \cdot G_{l_1,l_2} \right) \tag{6.4}$$

The **calibration cost** C_c reflects the time and effort it takes to conduct sensor calibration while staying at the spots. The cost $C_{c,i}[\omega]$ of a specific calibrator m_i in iteration ω can be computed based on a given selection of sensors $\mathbf{\Gamma}[\omega]$. We indeed know the calibration protocol (and thus duration) that needs to performed per sensor type. We further assume that the calibration that happens at the same spot is done in parallel. Thus, the **calibration time** that m_i spends at spot ν_l equals the maximum τ_k of all selected sensors at that spot (i.e. $\Gamma_{j,k}$=1 and $D_{j,l}$=1). Then $C_{c,i}[\omega]$ equals the sum of the calibration times at all the spots assigned to m_i:

$$C_{c,i}[\omega] = \sum_{l=1}^{L} \left(\max_{j,k} \left(D_{j,l} \cdot \Gamma_{j,k}[\omega] \cdot \tau_k \right) \cdot \sum_{l'=1}^{L} W_{l,l',i}[\omega] \right) \tag{6.5}$$

The **work load** of any calibrator m_i at iteration ω is the sum of his/her movement and calibration time:

$$C_i[\omega] = C_{w,i}[\omega] + C_{c,i}[\omega] \tag{6.6}$$

For any iteration, we assume that the maximum work load of any calibrator is $\hat{c}$, i.e. $C_i[\omega] \leqslant \hat{c}$,

$\forall(i, \omega)$.

The **total cost** of any iteration ω, denoted $C[\omega]$, is the weighted sum of: (a) C_{it} -the constant iteration overhead-, (b) $C_w[\omega]$ -the total movement time of all mobile calibrators-, and (c) $C_c[\omega]$ -the total calibration time at all spots with selected sensors-, i.e.:

$$C[\omega] = \mu_0 \cdot C_{\text{it}} + \mu_w \cdot C_w[\omega] + \mu_c \cdot C_c[\omega] \tag{6.7}$$

where $C_w[\omega] = \sum_i C_{w,i}[\omega]$ and $C_c[\omega] = \sum_i C_{c,i}[\omega]$. Since we assume that each spot is only visited once by one calibrator, the total calibration cost is computed directly from $\mathbf{\Gamma}[\omega]$, i.e.:

$$C_c[\omega] = \sum_{l=1}^{L} \max_{j,k} \left(D_{j,l} \cdot \Gamma_{j,k}[\omega] \cdot \tau_k \right) \tag{6.8}$$

6.3.3 Problem Formulation

We now introduce the **multi-sensor calibration planning** problem to minimize the **average cost** of operation over the maintenance period T. The problem is formulated as follows: Given the time span T, the map $\mathbf{G}$, the location matrix $\mathbf{D}$ and the sensor presence matrix $\mathbf{Q}$ of all the nodes, the calibration time τ_k and the calibration period T_k of all the sensor types, and the initial TTNC matrix $\mathbf{F}[1\_]$; find the total number of iterations Ω, and for each iteration $\omega = 1, 2, ..., \Omega$, find the time t_ω it takes place, the sensor selection $\mathbf{\Gamma}[\omega]$, and the number and the paths of calibrations $\{\mathbf{W}[\omega]\}$; such that the **average cost** of all iterations

over the time span T is minimized:

$$\min \ \frac{1}{T} \cdot \sum_{\omega=1}^{\Omega} C(\mathbf{\Gamma}[\omega], \{\mathbf{W}[\omega]\}) \tag{6.9}$$

$$\text{s.t. } t_1 = 0$$

$$\Gamma_{j,k}[\omega] \in \{0,1\}, \ \ \forall\omega, \forall j, \forall k$$

$$W_{l_1,l_2,i}[\omega] \in \{0,1\}, \ \ \forall\omega, \forall i, \forall(l_1,l_2)$$

$$F_{j,k}[\omega_-] \geqslant 0, \ \ \forall\omega, \forall j, \forall k$$

$$F_{j,k}[\omega_+] > 0, \ \ \forall\omega, \forall j, \forall k$$

$$\Gamma_{j,k}[\omega] \leqslant Q_{j,k}, \ \ \forall\omega, \forall j, \forall k$$

$$\mathbf{F}[\omega_+] = \mathbf{\Gamma}[\omega] \circ \mathbf{T}_N + (1 - \mathbf{\Gamma}[\omega]) \circ \mathbf{F}[\omega_-], \ \ \forall\omega$$

$$\mathbf{F}[(\omega{+}1)_-] = \mathbf{F}[\omega_+] - \min \mathbf{F}[\omega_+], \ \ \forall\omega$$

$$t_{\omega+1} = t_\omega + \min \mathbf{F}[\omega_+], \ \ \forall\omega$$

$$t_\Omega + \min \mathbf{F}[\Omega_+] \geqslant T \tag{6.10}$$

$$h_l[\omega] = \bigvee_{j=1}^{N} \bigvee_{k=1}^{K} \Gamma_{j,k}[\omega] \cdot D_{j,l}, \ \ \forall\omega, \forall l \tag{6.11}$$

$$\sum_{i=1}^{M} \sum_{l=1}^{L} W_{l,l_0,i}[\omega] = h_{l_0}, \ \ l_0{=}2, 3, \ldots, L, \forall\omega \tag{6.12}$$

$$C_i(\mathbf{\Gamma}[\omega], \mathbf{W}_i[\omega]) \leqslant \hat{c}, \ \ \forall\omega, \forall i \tag{6.13}$$

$\{\mathbf{W}[\omega]\}$ are valid path(s): constraints in §6.4.2 apply.

where $C[\omega]$ is the total cost of iteration ω given by Equation (6.7), which depends on the sensor selection $\mathbf{\Gamma}$ and the calibrators' paths $\{\mathbf{W}[\omega]\}$, i.e. $C[\omega]{=}C(\mathbf{\Gamma}[\omega], \{\mathbf{W}[\omega]\})$; obviously, it also depends on problem inputs (i.e. $\mathbf{G}$, $\mathbf{D}$, $\mathbf{Q}$, etc.) which are hidden for cleaner expressions. Unnumbered constraints above are related to the definition of sensor selection and TTNC in §6.3.1. Constraint (6.10) says the iterations need to cover the entire time span of T; (6.11) and (6.12) make sure all spots with selected sensors in $\mathbf{\Gamma}$ are visited in $\{\mathbf{W}\}$; (6.13)

says no calibrator should work for longer than $\hat{c}$ in any iteration. Additional constraints apply to ensure $\{\mathbf{W}\}$ are valid path(s) (§6.4.2).

The sensor calibration planning problem is NP-hard. It tries to minimize the total cost of all iterations while the choices of early iterations can affect and limit the choices of later ones. Also, the cost of each iteration $C[\omega]$ involves a **movement time** $C_w[\omega]$, which also needs to be minimized, and thus requires an optimization on the paths of the calibrators, which is a variant of the Multiple Travelling Salesman Problem (mTSP) or of the Vehicle Routing Problem (VRP) that are known to be NP-hard.

6.4 Solutions and Algorithms

Our formulation in Equations (6.9–6.13) suggests we find $\mathbf{\Gamma}[\omega]$ (sensor selection) and $\{\mathbf{W}[\omega]\}$ (path plan) simultaneously for all iterations. However, we observe the fact that if we know which spots the calibrators need to visit, we can optimize the paths to visit them accordingly. Hence, instead of attempting to minimize $\sum C/T$, for each iteration ω, we attempt a two-phase local optimization on the **single-iteration average cost**, $C[\omega]/(t_{\omega+1}-t_\omega)$, where we decouple the optimization of $\mathbf{\Gamma}[\omega]$ and $\{\mathbf{W}[\omega]\}$. Accordingly, for each iteration we have a **sensor selection planning** phase and a **multi-path planning phase**. In the selection planning phase, given the initial TTNC matrix $\mathbf{F}[\omega_-]$, we optimize the sensor selection $\mathbf{\Gamma}$, from which we derive the set of selected spots $H=\{h_l \mid h_l=1, \forall l\}$, which is then used in the path planning phase to decide the number of calibrators and the optimal path(s) to visit the selected spots.

Algorithm 3 TTNI-driven single-iteration local optimization algorithm for the sensor selection planning problem.

function solveSSPSingle ($\mathbf{G}$, $\mathbf{D}$, $\boldsymbol{\tau}$, $\mathbf{T}$, $\mathbf{Q}$, $\mathbf{F}$, $\boldsymbol{\mu}$, $\hat{c}$)

Input : $\boldsymbol{\tau}$ – $[\tau_k]$; $\mathbf{T}$ – $[T_k]$ k=1, ..., K;
$\boldsymbol{\mu}$ – $\{\mu_0, \mu_w, \mu_c\}$; Refer to §6.3.1, §6.3.3 for other symbols.

Output: $\boldsymbol{\Gamma}$ – Sensor selection matrix.

```
1  tMin ← min{F_{j,k} | Q_{j,k}=1 ∧ F_{j,k}>0}
2  tMax ← min T_k
3  tCandSet ← {F_{j,k} | tMin⩽F_{j,k}⩽tMax}
4  Initialize minCostAvg ← +∞ ; minGamma ← null
5  for each T_cand in tCandSet do
6      Γ ← 0_{N×K}
7      for j in 1, ..., N ; k in 1, ..., K do
8          Γ_{j,k} ← (Q_{j,k}=1 ∧ F_{j,k}<T_cand
9          for j' in 1, ..., N ; k' in 1, ..., K do
10             if D_{j,l}=D_{j',l}, ∀l then
11                 Γ_{j',k'} ← (Q_{j',k'}=1 ∧ τ_{k'}⩽τ_k)
12     H ← {l | ∃(j,k) s.t. Γ_{j,k}=1 ∧ D_{j,l}=1} ; β ← 0_L
13     for l in 1, ..., L do β_l ← max_{j,k}(D_{j,l}·Γ_{j,k}·τ_k) ;
14     cost ← μ_0·C_it + μ_c·C_c(D, Γ, τ) + μ_w·C_w(G, solveMPPGreedy(G, ĉ, H, β))
15     if (costAvg ← cost/T_cand) < minCostAvg then
16         minCostAvg ← costAvg ; minGamma ← Γ
17 return Γ ← minGamma
```

6.4.1 Sensor Selection Planning Algorithms

Leveraging the discrete nature of TTNC and the definition of TTNI (time to next iteration), we propose the TTNI-driven local optimization algorithm. The intuition behind this algorithm is to exhaust the possible values of TTNI (i.e. $t_{\omega+1}-t_\omega$) and find the "cheapest" one to fulfill.

The procedure of the TTNI-driven local optimization is shown in Algorithm 3. It involves the following steps: (1) Determine **all the possible values of TTNI** that could result from any possible sensor selection in this iteration. The minimum TTNI candidate is $\min\{F_{j,k}[\omega_-] \mid Q_{j,k}=1 \wedge F_{j,k}[\omega_-]>0\}$, selecting only the sensors that need immediate calibration (Ln 1). The maximum TTNI candidate is $\min T_k$, selecting **all sensors** (Ln 2).

All values in $\mathbf{F}[\omega_-]$ between them become TTNI candidates (Ln 3). In the worst case, the number of TTNI candidates is $O(N{\cdot}K)$ (2) For each TTNI candidate T_{cand}, tentatively assume it to be the desired TTNI and create the minimum selection of sensors to meet the TTNI, i.e. let $\Gamma_{j,k}$=1 if $Q_{j,k}$=1 and $F_{j,k}[\omega_-]<T_{\text{cand}}$ (Ln 5–8); then add all sensors that are co-located with the selected sensors and that do not induce extra time for calibration (Ln 9–11, because their calibration is done in parallel, if it takes a shorter time). Generating $\mathbf{\Gamma}$ from T_{cand} takes $O(N{\cdot}K{+}N{\cdot}L)$ time. Compute the single-iteration average cost from $\mathbf{\Gamma}$ (Ln 12–14). (3) Select the TTNI candidate that gives the minimum average cost (Ln 16), and its corresponding $\mathbf{\Gamma}$ is the output of the algorithm. The worst-case running time excluding the time used to compute or estimate the movement time, is $O(N^2{\cdot}K^2{+}N^2{\cdot}K{\cdot}L)$.

If during step (2) we are able to compute the optimal paths of calibrators, we will compute the best cost evaluation for each selection and find the local optima. Unfortunately, multi-path planning is also NP-hard. We then propose two heuristics: a fast nearest-neighbor-based greedy algorithm, and an improved genetic algorithm (GA). During sensor selection, we use the faster greedy algorithm to estimate the movement cost; once the selection is done, we use GA to generate the final path(s) for the iteration.

6.4.2 Multiple-Path Planning Algorithms

The **multi-path planning problem** for a specific iteration ω refers to a sub-problem in our two-phase local optimization solution to the sensor calibration planning problem. The objective is to generate a set of paths $\{\mathbf{W}[\omega]\}$ of minimum cost (i.e. movement time $C_w[\omega]$) for the selected spots yielded by the sensor selection $\mathbf{\Gamma}[\omega]$. It is a variant of the classic mTSP or VRP: we determine the number of calibrators based on the demand instead of having the number m of travellers given, as in mTSP. Also, evaluating the calibrator workload constraint involves the movement time of individual calibrators, which adds to the

complexity of solutions.

Hence, we derive the following mixed-integer-programming (MIP) formulation of the multi-path planning problem based on a flow-based three-index MIP formulation of mTSP [10], adding appropriate modifications to match our assumptions and constraints: Given a map $\mathbf{G}$, the location of the nodes $\mathbf{D}$, the sensor selection $\mathbf{\Gamma}$, and the calibration time τ_k, $\forall k$; find $\mathbf{W}_{L\times L\times M}$ and helper variables $\mathbf{U}_{L\times M}$ to

$$\min \sum_{l_1=1}^{L}\sum_{l_2=1}^{L}\left(G_{l_1,l_2}\cdot\sum_{i=1}^{M}W_{l_1,l_2,i}\right) \tag{6.14}$$

$$\text{s.t. } W_{l_1,l_2,i}\in\{0,1\},\ \ \forall i,\forall(l_1,l_2)$$

$$W_{l,l,i}=0,\ \ \forall l{=}2,3,\ldots,L,\forall i$$

$$\textstyle\sum_{l_2=1}^{L}W_{1,l_2,i}=1,\ \ \forall i$$

$$\textstyle\sum_{l_1=1}^{L}W_{l_1,l,i}-\sum_{l_2=1}^{L}W_{l,l_2,i}=0,\ \ \forall i,\forall l$$

$$\textstyle\sum_{l_1=1}^{L}\sum_{i=1}^{L}W_{l_1,l_2,i}=h_l,\ \ \forall l_2 \tag{6.15}$$

$$u_{l,i}\geqslant 2,\ \ \forall i,\forall l$$

$$u_{l_1,i}-u_{l_1,i}+1-(L-1)\cdot(1-W_{l_1,l_2,i})\leqslant 0,\ \ \forall i,\forall(l_1,l_2)$$

$$\textstyle\sum_{l_1=1}^{L}\sum_{l_2=1}^{L}W_{l_1,l_2,i}\cdot\big(G_{l_1,l_2}+\Upsilon_{l_2}\big)\leqslant\hat{c},\ \ \forall i \tag{6.16}$$

Where $W_{l_1,l_2,i}$=1 if calibrator m_i visits spot ν_{l_2} immediately after spot ν_{l_1}; or 0 otherwise. $\{\mathbf{W}\}$ is represented in a more general form that could contain calibrators with no assignment (i.e. that are not dispatched). M is the maximum number of calibrators; assuming we always have enough calibrators, L would be an effective upper bound of M to be used in solvers. Unnumbered constraints are related to the construction of multiple valid paths. Constraint (6.15) makes sure all selected spots are visited by exactly one calibrator; (6.16) enforces the maximum workload of calibrators, where Υ_l is the total calibration time spent at spot ν_l, i.e. $\Upsilon_l[\omega]=\max_{j,k}(D_{j,l}\cdot\Gamma_{j,k}[\omega]\cdot\tau_k)$.

This formulation allows us to apply MIP solvers directly. However, the problem is NP-hard; the number of independent variables and the number of constraints in this MIP formulation are both in the order of $O(L^3)$, resulting in a huge solution space. It is hard for any MIP solver to optimally solve the problem in a reasonable amount of time using the available hardware [10]. In particular, we tried two widely used solvers: GLPK (GNU Linear Programming Kit, open-source [75]) and Gurobi (commercial software [77]). None of the two solved the problem in less than 48 hours for $L \geqslant 15$. To solve the problem at larger scale, we propose two heuristics: a greedy algorithm derived from the nearest neighbor heuristic of traditional TSP, and an improved genetic algorithm (GA) based on the one proposed by Sedighpour, et al. [103] for mTSP. For a clean design of the algorithms, after the completion of the sensor selection planning phase, the planning framework computes the set of **selected spots** H and the calibration time $\boldsymbol{\beta}$ at these spots from the sensor selection matrix $\boldsymbol{\Gamma}$, the deployment matrix $\mathbf{D}$, and the calibration time τ_k of the sensor types. Note that the depot is not a "selected spot" ($\nu_1 \notin H$) because we do not deploy any node at the depot, though it exists in the actual paths of all mobile calibrators as the departure and destination.

Nearest-Neighbor-Based Greedy Heuristic

The nearest neighbor algorithm for TSP starts with a tour containing only one spot. At each step, it determines that the next spot to visit as the one that is closest to the last visited spot, and loops until all the spots are visited.

Inspired by this straightforward TSP algorithm, we derive our greedy algorithm for the multi-path planning problem shown in Algorithm 4 as follows: (1) Start with a set of empty paths (i.e. all the calibrators stay at the depot) and the set of all selected spots H. (2) At each step, for every unvisited spot, compute the extra travel and calibration time yield by adding it to the end of the path of each mobile calibrator as long as the calibrator is not overloaded. Pick the spot-calibrator pair that induces the least additional movement time.

Algorithm 4 Nearest-neighbor-based greedy algorithm for the multi-path planning problem.

function solveMPPGreedy (**G**, $\hat{c}$, H, $\boldsymbol{\beta}$)

Input : **G** – Map; $\hat{c}$ – Maximum workload;
H – Set of selected spots;
$\boldsymbol{\beta}$ – Vector of calibration time at selected spots.

Output: {**W**} – Set of paths.

```
1  Initialize pathSet ← [ ]
2  while H is not empty do
3      minInc ← +∞ ; minSp ← minPath ← null
4      for each path in pathSet ; last ← path[−1] do
5          oldTime ← getMoveTime (path) + Σ_{l∈path} β_l
6          for each sp in H do
7              dCw ← G[last, sp] + G[sp, 1] − G[last, 1]
8              if oldTime + dCw +β_sp ⩽ ĉ then
9                  if dCw < minInc then
10                     minInc ← dCw
11                     minSp ← sp ; minPath ← path
12     if minInc is finite then minPath.append (minSp) ;
13     else
14         for each sp in H do
15             if (dCw ← G[1, sp] + G[sp, 1]) < minInc then
16                 minInc ← dCw ; minSp ← sp
17         newPath ← [minSp] ; pathSet.add (newPath)
18     H.del (minSp)
19 return {W} ← convertPathVecToMatrix (pathSet)
```

Note that the spot could be added to an old calibrator (Ln 4–12) or a new calibrator (Ln 14–17). (3) Loop until all the spots are visited (Ln 2, 18). Note: Our actual implementation of this algorithm caches the movement and calibration time associated with each calibrator to reduce redundant computation, so the worst-case running time of this algorithm is $O(L^3)$.

Improved Genetic Algorithm (GA)

We design our genetic algorithm (GA) based on the mTSP GA solution of [103]. Features are added to address the peculiarities of our MPP formulation, i.e. the variable number of

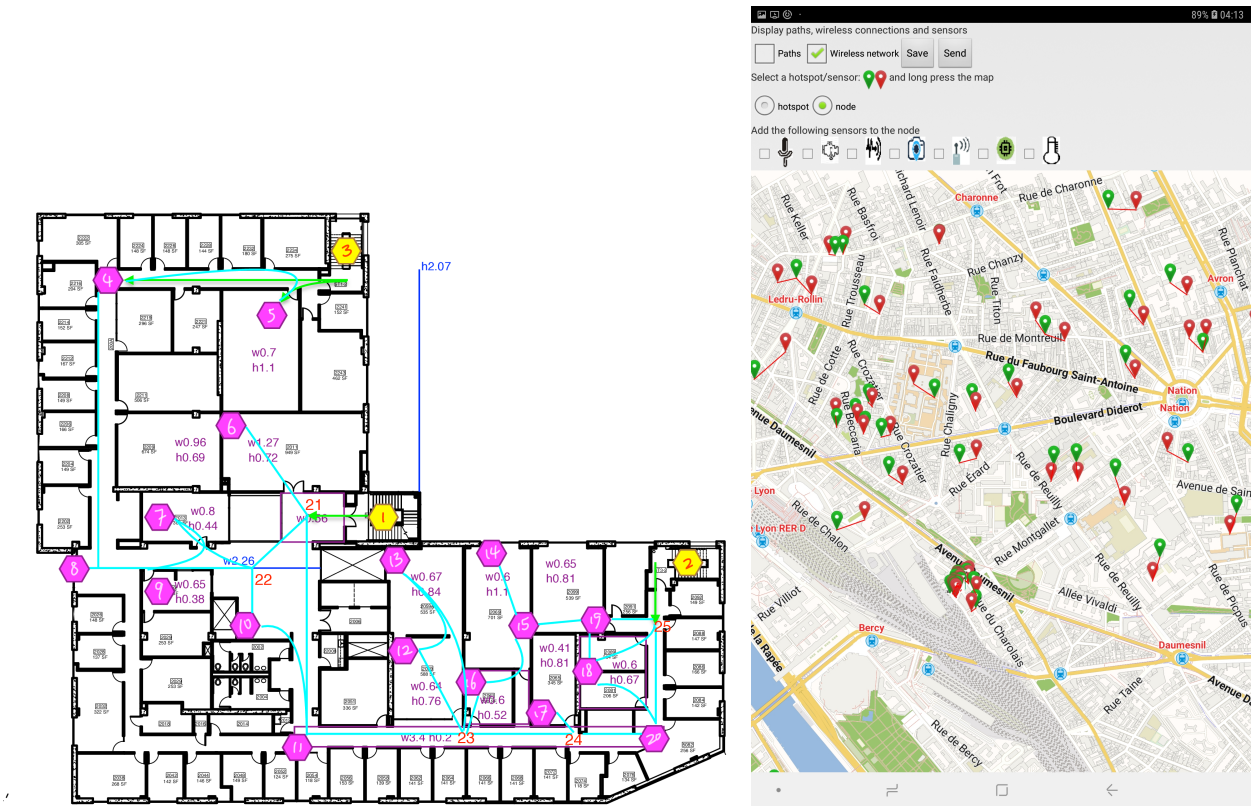
calibrators, the workload constraint, and the map represented by a directed-graph.

A **chromosome** is an integer vector that is made of two parts: a permutation of all selected spots (1st half) and an assignment mapping the spots to mobile calibrators (2nd half). If the number of selected spots is $|H|=L'$, a chromosome will have length $2L'$. The assignment (2nd half) is represented by the number of spots visited by each calibrator, so these integers should all be in range $[0, L']$ and sum up to L'. For example, chromosome $[2, 4, 5, 6, 3, 2, 3, 0, 0, 0]$ means $L'=5$ and that m_1 will visit spots ν_2, ν_4, and m_2 will visit ν_5, ν_6, ν_3. The **fitness** is the negative of the total movement time of all mobile calibrators, and the **selection** is done by a standard scaled-fitness proportional selection.

The **initial population** is composed of randomly generated individuals. The permutation is performed by a uniformly random permutation generator, and the assignment is done by uniformly and randomly picking an integer and subtracting it from the total number of selected spots until none is left. The **crossover** is done by applying a standard "order crossover" on the first half of the chromosome.

Because of the variable number of mobile calibrators, we design three helper functions that apply to chromosomes: (a) **compress**: shift all zeros in the assignment section to the end and non-zeros values to the beginning; (b) **split**: check if any assignment ($\geqslant$2 spots) leads to an overloaded calibrator, randomly split it into two calibrators, and loop until none is found; (c) **merge**: check if there exists a pair of assignments that can be merged into one without overloading the calibrator; then merge the first pair found. Among the three, **compress** and **split** are applied to every newly-generated chromosome during population initialization, mutation, and crossover, while **merge** is applied as one type of mutation.

Apart from "merge", there are three other types of **mutation**: (a) two-point swap, (b) segment reversal, and (c) 3-opt local optimization. Every time a mutation is triggered, we randomly pick one of the four types of mutation functions. (a) and (b) are straightforward.



(a) DBH 2nd floor; data set contains all six floors; real and synthetic spots.

(b) Paris area of 10 km^2; synthetic spots.

Figure 6.3: Smart space structure and spot location used in evaluation.

3-opt [17] is a local optimization for TSP, which tries to break a tour into three segments by removing three edges, and reconnect the three segments into a new but shorter tour. 2-opt is a commonly used local optimization for TSP on undirected graphs, but an odd numbered opt is required for digraphs to avoid reversing any segment, which makes it faster to compute the new movement time.

Finally, the tunable parameters such as the population size, elite-keeping size, and the termination conditions are assigned by the framework according to the problem size (i.e. L').

Table 6.1: Experimental setup for the performance evaluation.

Input Data			Indoor		Outdoor
			Normal	**Emergency**	
Map	L	Num. of Spots	60		63
	G	Pairwise Dist.[a]	$\leqslant$53 sec		$\leqslant$73 min
Sensor	K	Num. of Types	10		8
	τ	Calib. Time	1–30 min		0.25–1 min
	T	Calib. Period	14–91 d	7–91 d	28–123 d
Nodes	N	Num. of Nodes	100 (varies if independent variable)		
		Sensor Presence	50 (varies if independent variable)		
User Req.	$\hat{c}$	Max. Workload	2 hours		4 hours
	$\boldsymbol{\mu}$	Coefficients	C_{it}=10000, μ_0=1 μ_w=5, μ_c=1		
	T	Maintenance P.	360 days		

[a] Pairwise distance correspond to the shortest traveling time.

6.5 Validation

We evaluate the performance of our proposed multi-sensor calibration planner (§6.5.1) using realistic data derived from testbeds and present the steps we are taking towards a usable system for calibration planning (§6.5.2). This includes modules to facilitate the modeling of the (indoor/outdoor) environment and that is intended to provide navigation guidance to the calibrators (via an Android app).

6.5.1 Performance Evaluation and Results

We conduct a series of evaluations using three sets of input data. The two first involve the instrumented building at UC Irvine (Figure 6.3a) we discussed in §6.2, which is used for everyday monitoring (normal condition) and for supporting emergency operations when needed. The desired calibration frequency is a parameter that is learned from empirical

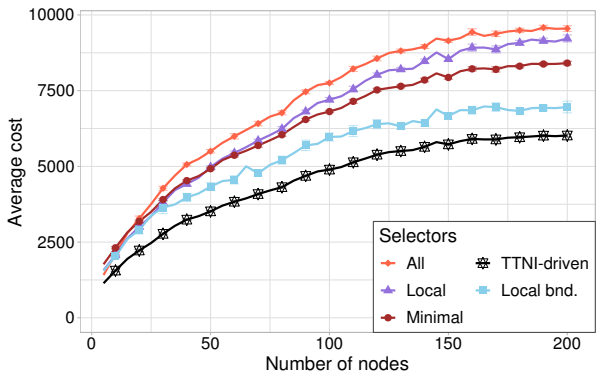
study and that depends on the context. We consider the actual deployment in our real testbeds and generate additional spots (with sensors and nodes) using a similar pattern. Using the service we present in §6.5.2, we synthesized the third data set that relates to an outdoor urban environment in which sensors are placed to monitor noise and air quality (Figure 6.3b). The parameters are summarized in Table 6.1.

Experimental Setup

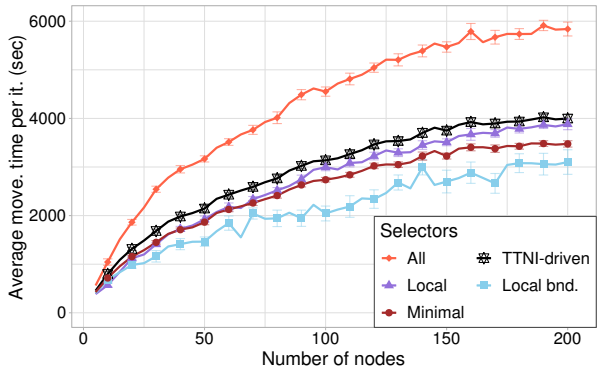
We compare our sensor selection with two naïve sensor selection strategies that aim at "always selecting *all* sensors" (regardless of their TTNC) and "only selecting the *minimal* set of sensors" (i.e. those we must calibrate because their TTNC reaches 0). In addition, we also investigate two simple selection strategies. The former consists in selecting all sensors that are co-located with the sensors that form the minimal set ("*local*"). The latter consists in only selecting the sensors that can automatically calibrate with each other without human intervention and that henceforth do not induce additional calibration time ("*local bounded*"). For multi-path planning, we evaluate the performance of two MIP solvers (*GLPK* and *Gurobi*), our two path planning heuristics (NN-based *greedy* and *GA*), and a naïve strategy that sends one calibrator to *each* spot and that should give the highest cost. Algorithm running time is evaluated on the OpenLab cluster of Dept. Computer Science at UCI, where each computing node has 2x Quad-core Intel Xeon 3.0GHz CPU E5450 CPUs.

Indoor vs. Outdoor Results

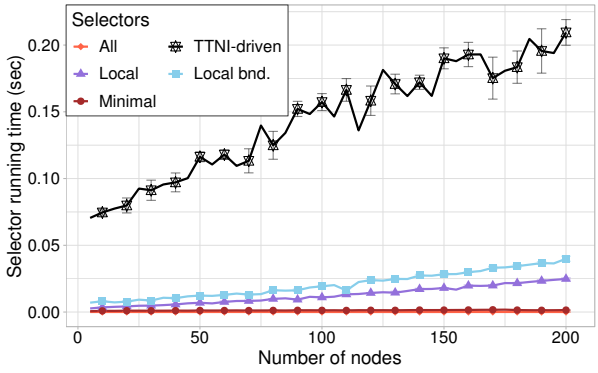
In an indoor environment (Figure 6.4), our algorithm (TTNI-driven sensor selection and GA-based multi-path planning) always result in a lower average cost for N ranging from 5 to 200. Compared to the naïve sensor selection strategies, such as "selecting **all** sensors" and "selecting the **minimal** set of sensors" (still considering GA-based multi-path planning), our



(a) Average cost vs. number of nodes.



(b) Average moving time (sec) per iteration vs. number of nodes.

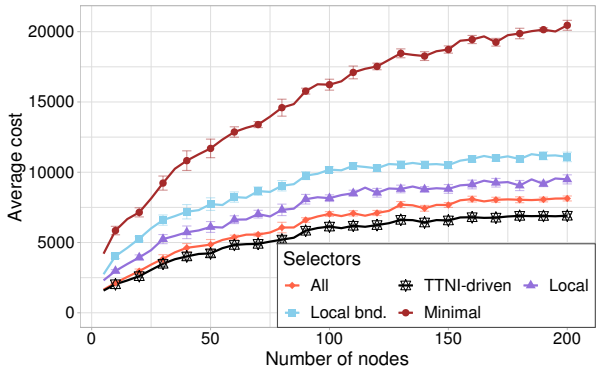


(c) Mean running time (sec) vs. number of nodes.

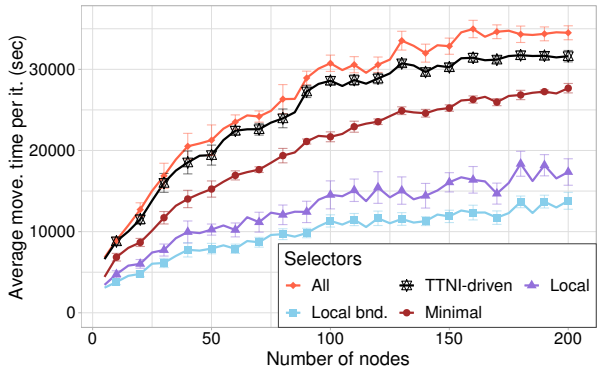
Figure 6.4: Impact of the number of nodes on the sensor selection algorithms in an indoor setup, with sensor presence rate $\sum \mathbf{Q}/(N \cdot K)$=0.5.

algorithm combination provides up-to 30% improvement in the long-term average cost. Note that even though our algorithm does not always end up with the lowest cost per iteration (Figure 6.4b), it makes a fair trade-off between the cost and the time (between iterations). Figure 6.4c shows that the time spent to select sensors is short – less than 1 sec for a reasonably complex building incorporating 200 nodes and 60 spots.

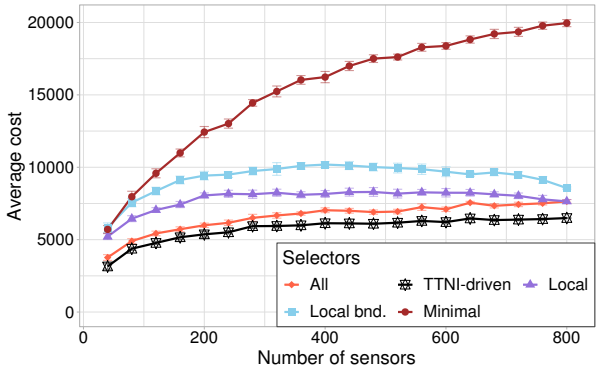
The same trend also applies in the outdoor environment (Figure 6.5), where the distances between spots are significantly longer (Table 6.1). As the spatial span of the setup grows, the difference among the algorithms becomes more dramatic (note the different y-scale in Figures 6.4a,6.4b and 6.5a,6.5b). Certain naïve approaches are very sensitive to this change (Figure 6.5b, "minimal" and "local bounded"), while our algorithm shows stable performance



(a) Average cost vs. numb. of sensors N; sensor presence $\sum \mathbf{Q}/(N{\cdot}K)$=0.5; outdoor scenario.



(b) Average movement time (sec) per iteration vs. N; $\sum \mathbf{Q}/(N{\cdot}K)$=0.5; outdoor scenario.



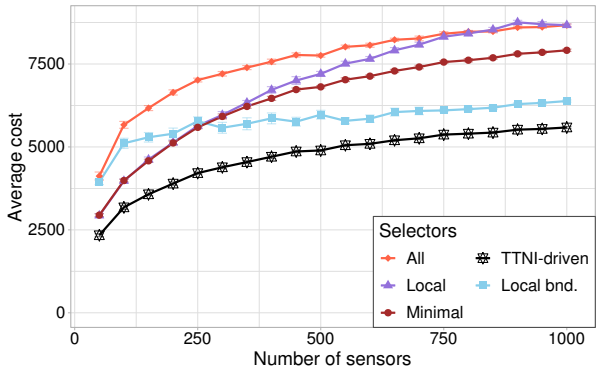
(c) Average cost vs. number of sensors $\sum \mathbf{Q}$; N=100; outdoor scenario.

Figure 6.5: Evaluation of the sensor selection algorithms in an outdoor scenario.

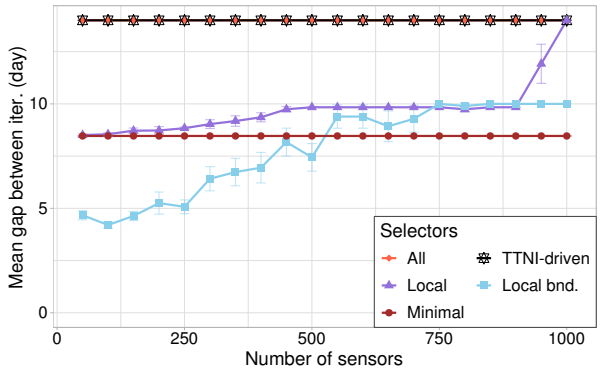
in both indoor and outdoor settings.

Normal vs. Emergency Condition Results

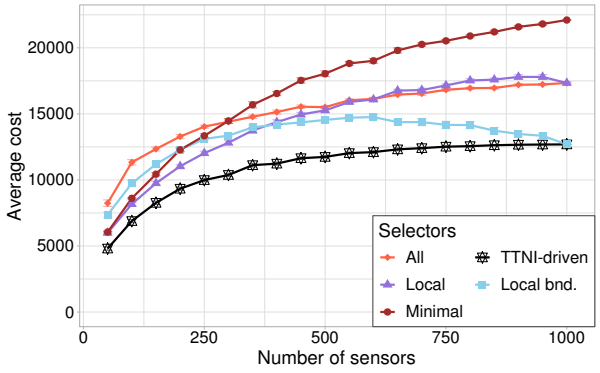
Having demonstrated the effectiveness of our algorithm in indoor and outdoor settings, we further study the performance of our approach in an emergency scenario where the calibration requirements of certain sensor types are increased (Figures 6.6a and 6.6c, note the difference in y-scale). When calibration is required more frequently for some sensors, the naïve/simple approaches suffer from a big increase in average cost, especially when the sensors are deployed densely ($\sum \mathbf{Q}/(N{\cdot}K)$>0.5), while the performance of our algorithm and "local bounded" are less affected. We also notice that unlike the simpler approaches ("local bounded" or



(a) Average cost vs. number of sensors $\sum \mathbf{Q}$; normal scenario.



(b) Mean interval (day) between iterations vs. number of sensors; normal condition.



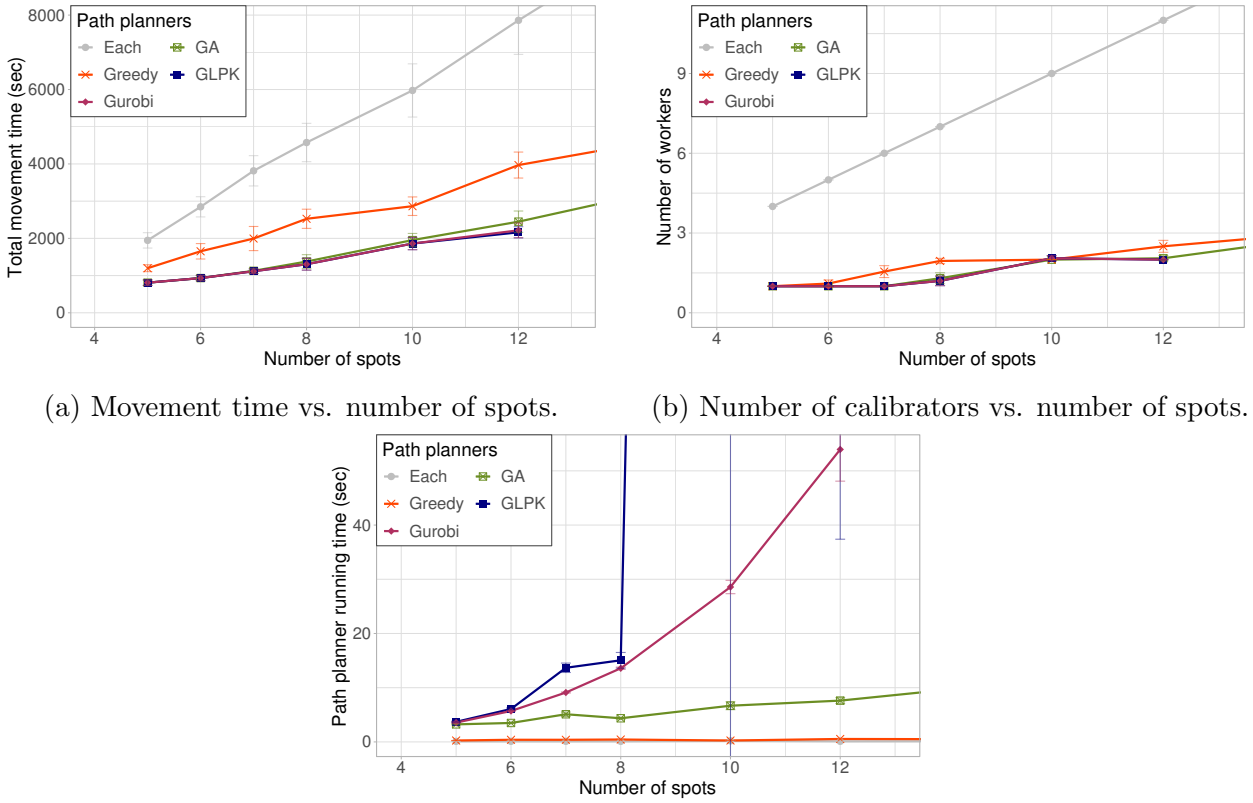
(c) Average cost vs. number of sensors; emergency condition.

Figure 6.6: Impact of the number of sensors $\sum \mathbf{Q}$ on the sensor selection in an indoor environment containing 100 nodes.

"minimal"), our TTNI-driven sensor selection algorithm avoids the desynchronization of the periodical calibrations, while the "local bounded" strategy does so with a small number of sensors (Figure 6.6b).

Scalability Results for the Multi-Path Planning

Figure 6.7 compares the performance of multi-path planning algorithms for the number of spots $L \leqslant 12$: GA provides close-to-optimal solutions but takes 8–10 sec for L=12 (20–60 sec for L=60); the greedy heuristic is much faster (approx. 0.5 sec for L=60), which makes it suitable as an estimator during the sensor selection planning. GLPK and Gurobi (i.e. the



(a) Movement time vs. number of spots.

(b) Number of calibrators vs. number of spots.

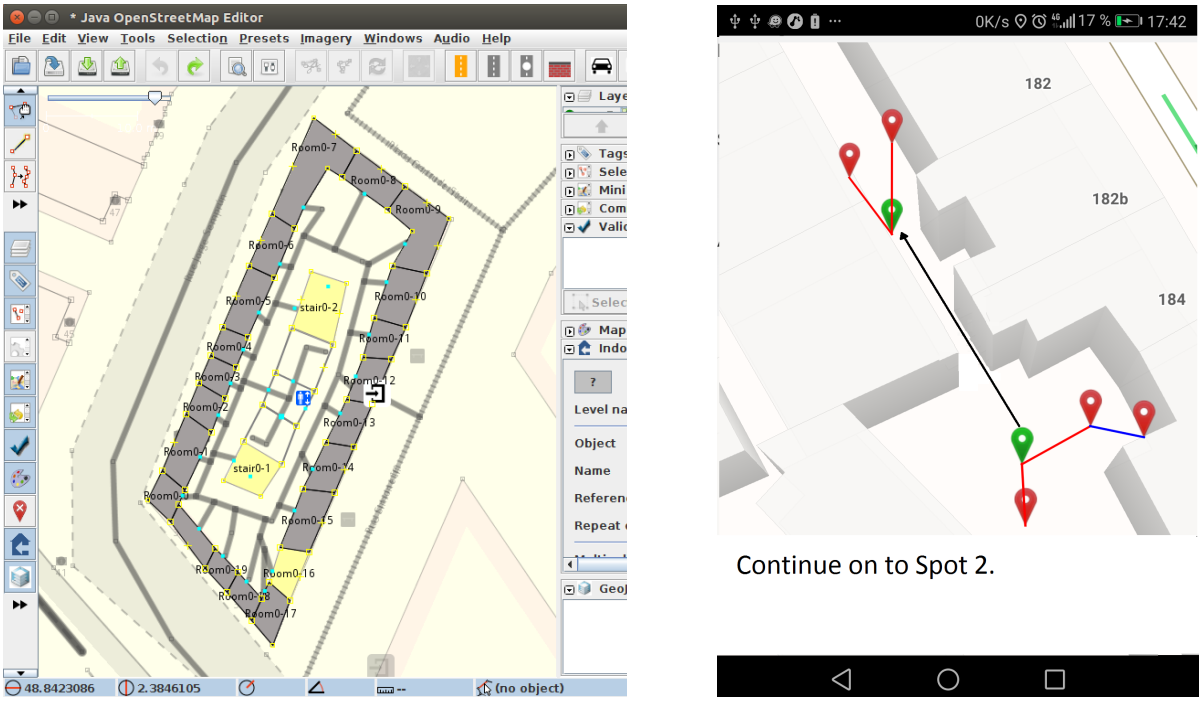
(c) Running time (sec) vs. number of spots.

Figure 6.7: Scalability of the multi-path planning solvers and proposed heuristic algorithms.

MIP solvers) could not terminate within 48 hours for $L \geqslant 15$ so we aborted.

6.5.2 Toward a Usable System

Creating a calibration planning service is not trivial. In addition to the algorithm for planning, we are implementing a service to enable flexible calibration. To support usability, we have implemented a toolkit that serves to generate 3-dimensional maps and provides navigation guidance to the mobile calibrators. We create the 3-D navigable maps using OpenStreetMap (www.openstreetmap.org), which provides a basemap of the outdoor environment, e.g., roads (lines), junctions (points), buildings (polygons), relations (groups). The basemap is further edited (Figure 6.8a) so as to (i) detail the inner structure of the buildings of interest (if any), and (ii) layer a number of features including the paths accessible in the



(a) Modeling indoor space of a building using JOSM. (b) Navigation guidance.

Figure 6.8: Mapping of an indoor/outdoor environment. Spots (green markers) are placed; a set of nodes (red markers) compose a spot (red line). Nodes can be connected by a wireless link (blue line).

indoor environment. To design the inner structure of the building, we rely on JOSM [80] that leverages the indoor model promoted by OpenStreetMap [82, 78]. Following, we extract a representation of the connection graph from the resulting map using GraphHoper [76]. This graph is composed of edges corresponding to streets, roads as well as indoor-pedestrian paths, and nodes representing junctions.

The next step consists in placing on the map, the spots that should be visited along with the nodes and the related sensors. The wireless connections among sensing nodes are also modeled to inform the propagation of calibration parameters. The map and the (generated) connection graph are exploited by the Android app that we implemented for assisting the mobile workers (Figure 6.8b). The app computes the shortest path (avoiding obstacles) between any 2 spots and provides this information to the multi-sensor calibration planner. For instance, with 60 spots and 77 nodes equipped by 188 sensors mapped over an urban area

of 10 km^2 (Figure 6.3b), the resulting connection graph is generated in 2.967 s. This graph, which contains 8768 edges (e.g., streets) and 5661 vertices (e.g., junctions), is further used to compute the paths between any two spots; such a computation takes in average 0.0195 s for an average distance separating two spots of 1387 meters. Relying on the multi-sensor calibration planner, the mobile app further provides navigation guidance to the calibrators (Figure 6.8b).

6.6 Summary and Discussion

Networked and embedded devices that sense from and act on the environment are increasingly employed as part of IoT applications to blend our surrounding physical world with the digital world and facilitate human-machine interactions. Nevertheless, the effective deployment of IoT remains a challenging task with a multitude of pitfalls: once deployed, sensors are subject to drifts, bias, and unexpected errors and thereby fail to provide the expected, meaningful actionable data. Cost-effectively planning the on-site calibration of IoT devices can help in the sustainable long term operation of deployments.

Our work focuses on the collaborative and distributed maintenance of an IoT-based system. We build on our experiences in deploying sensors over smart spaces and on addressing the calibration tasks to enhance the quality of the gathered observations. We frame the resulting multi-sensor calibration planning problem as an optimization problem wherein the objective is to determine how many calibration iterations are necessary, which sensors should be calibrated at each of these iterations and the number of mobile calibrators (workers) that are required (as well as their respective calibration paths), such that the average cost of all iterations is minimized and under the constraint that the calibrators should not be overloaded. The proposed two-phase iterative local optimization approach first creates a selection of sensors for each iteration, and introduces new methods (mTSP variants, heuristics) to compute

a set of paths for the calibrators based on the selection. Our evaluation shows that the proposed algorithms solve the sensor calibration planning problem in an effective manner compared to naïve/simple solutions.

Introducing a service to address large scale calibration issues in IoT deployments is promising; while our initial studies demonstrate the value and feasibility of this approach, long-term studies in the field are required to adapt and fine-tune the calibration process for different types of applications and external dynamicity. An interesting aspect of such systems is that they are inherently reflective – deployed systems can help develop an understanding of how IoT data evolves which in turn can help adapt the systems that gather and process this data.

Chapter 7

Conclusion

The Internet of Things (IoT) is becoming prevalent in the realization of large-scale systems for smart communities and smart cities. The combination of in-situ and mobile deployments is a promising direction towards this goal.

In this thesis, we identify key data collection challenges and propose planning-based solutions to enable the efficient operation and maintenance of community-scale IoT deployments with both mobile and in-situ devices. These challenges were derived from observations and lessons learned both from our own deployment experiences and those of others. Our techniques leverage prior knowledge of application needs, community infrastructure, device heterogeneity, and data characteristics. They optimize the activities of the devices under data budgets and timeliness constraints to achieve a balance between data utility (i.e., accuracy, importance, and timeliness) and cost (i.e., that of deployment, operation, and maintenance). We explore and evaluate our solutions within the context of urban environmental sensing and address three major research problems regarding IoT data generation, data upload, and sensor calibration (i.e., maintenance), respectively. The effectiveness of our approach has been demonstrated through performance evaluation driven by real and synthetic traces. We

believe that in the future to come, the increasing prevalence and pervasiveness of IoT will highlight the values of our contributions.

7.1 Towards an End-to-End Integration

Currently, the planning framework for the three stages (i.e., data generation, data upload, and sensor calibration) work independently and strive to leverage any information that is available in the community context to improve the efficiency at respective stages – data generation planning and data upload planning regulate the everyday operation of community IoT systems; sensor calibration planning helps to create system maintenance plans in the long term. However, this functional independence does not preclude data sharing between the planning algorithms. For example, the crowd participants for whom we create data generation plans can also help measure/observe the dynamicity in the system or the environment, which can add to the prior knowledge we need for data upload planning. They may also opportunistically check/validate the low-cost in-situ sensors and report failures, which can trigger a dynamic maintenance/calibration cycle – our current model of sensor calibration planning can handle such dynamicity, though our solution and evaluation in §6 assume deterministic cases for simplicity.

Meanwhile, in addition to the three stages (i.e., data generation, data upload, and sensor calibration) that we primarily focus on in this thesis, there are many other stages in the community IoT workflow that could be improved through planning techniques that exploit data/knowledge available in communities and cities. For example, we can leverage the geographical structure, the network topology, and the information of registered devices in the communities for better planning and resource allocation in location-based routing, load balancing, and edge computing. In the IoT architecture, planning techniques are often implemented as applications that run on the cloud or the edge servers. They leverage the

IoT data exchange services for data access and sharing and could target either short-term operation or long-term maintenance. If applicable, elements of the light-weight dynamic adaptation logic can also execute on resource-constrained devices to enable timely response and adjustment. This brings the opportunity in determining how to partition the planning process in the network (i.e., cloud vs. edge). Future work focusing on an end-to-end planning system will help support better dynamic adaptation techniques, further integration of applications and scheduler modules, as well as an efficient data exchange mechanism that facilitates data sharing between them.

At the same time, a global cross-layer planning framework is a new direction but needs further research and investigation. Deeper cross-layer integration is necessary for deriving globally optimal solutions and plans. However, it could complicate the system design and increase the dependency on case-specific characteristics, which may cause the planning approaches to lose generality. It is also interesting to observe the tradeoff between generality and the effectiveness of planning. For example, mixed integer planning solvers and random search heuristics (e.g., GA) are general frameworks that could handle many practical optimization problems including ours, as long as the problem can be modeled correspondingly. However, our work shows that context-aware solutions often result in better effectiveness and efficiency.

7.2 Future Work: Phased Planning Techniques

In the core chapters (§4, §5, §6), we have demonstrated the effectiveness of our approaches to exploit the mobile plus in-situ deployments. Here, we briefly summarize the future directions of each core chapter in this thesis.

Data Generation Planning

We continue our work on data generation planning with: (a) processing real-world air quality data collected from an open data platform in Taiwan, based on which we derive more practical spatiotemporal impact functions, and (b) enabling improvement in planning algorithms to leverage look-ahead planning techniques that fit better into the temporal impact models.

The spatiotemporal scheduling framework and algorithm that we described in §4.3 have been tested in a community-scale setup that contains up to 200 nodes. To further scale up the system for large districts and cities, it will be necessary to have a scheduling hierarchy that allows us to offload some work to edge servers and devices. Note that the current solution assumes that the devices and networks always function as expected and that the plans can always be communicated and applied in real-time. This is out of scope for this thesis while we focus on the major challenges (e.g. the non-uniform deployment), but it may not be true in a real-world community IoT system. Thus, future work in data generation planning will also include a study on the impact of network connectivity uncertainties and a more comprehensive and fine-grained mobility model that enables better look-ahead planning. Further extension to this stage could also incorporate a model that allows prioritization of cells and time frames, which could be useful in emergencies.

Data Upload Planning

In the current data upload planning framework that we discussed in §5.2, we consider the planning for individual mobile data collectors (MDCs) independently. While this simplifies the problem and allows us to focus on challenges from non-uniform network availability and multiple types of dynamicity, it may not result in the optimal overall performance in situations where multiple MDCs may collaborate for the data collection in the same region. A joint data upload planning of multiple MDCs may better balance resource utilization.

Also, in our current model, we only apply this approach to MDCs with predetermined paths. A model that integrates with trajectory manipulation and path planning will allow more opportunistic participation and better dynamic adaptation in extreme conditions (e.g. emergencies). Further extension to this stage could also leverage a comprehensive multi-network cost-effectiveness model for the utilization of heterogeneous networks [90].

Sensor Calibration Planning

We are currently developing a navigation application for smartphones that can interface with the SCALE (or similar) system architecture. Besides, to capture system/device dynamicity, we need the design of a new set of solutions and experiments accordingly to enable on-demand calibration and maintenance based on field reports. Future efforts will also focus on a systematical study on the degradation and drift of common sensor types and long-term studies with an end-to-end system. For example, our team is seeking collaboration with AirUCI [71] to leverage their professional devices and existing deployments in such studies. This could help us gain a better understanding of the complexity of sensor calibration and device maintenance in IoT systems and derive more realistic experimental setups.

7.3 Future Directions: Mobile Plus In-Situ Community IoT

The future of IoT-enabled smart communities and cities will inevitably include larger-scale deployments, multi-networks, and heterogeneous mobile devices with different mobility models. Independent smart systems nowadays are also likely to become integrated at different levels in order to provide better service and better reliability in different conditions. Areas that are highly likely to become relevant to the topic of mobile plus in-situ deployments

include:

- **Smart traffic systems**: Such systems can provide useful traffic data for decision makers to better understand the overall traffic conditions and the correlation between traffic and other environmental metrics. At the same time, smart traffic systems can also leverage the data collected from other systems to enable proactive traffic regulations.
- **Vehicular networks**: Such systems include vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications. The vehicles provide relatively predictable mobility, which can be leveraged for data collection and routing. Being able to integrate with smart traffic systems also enables new applications (e.g. virtual traffic signs) that help improve the efficiency and throughput of traffic systems.
- **Autonomous vehicles and drones**: These autonomous devices could leverage data from other systems for a better vision of the environment in their planning mechanisms, while they can also provide opportunistic and planned mobility to help extend the sensing coverage and facilitate data collection.
- **Smart city infrastructures** (e.g. water distribution, drainage, power grid): These systems contain devices with different mobility models (e.g. the floating devices that travel with water in the pipes) and expose a different set of physical phenomena for monitoring. Applying our techniques to such systems is an exciting direction that could benefit the life quality of residents of smart cities and requires further research.

Our group has initiated the work on leveraging public transportation systems in data collection, which leads to interesting research problems on clustering, deployment, instrumentation planning, and data routing. We are also exploiting the use of drones for extending sensing coverage and modeling spatial attributes, which require further studies on mobility models, power consumption issues, and three-dimensional features of wireless network systems, etc.

While our research currently focuses on urban monitoring, we also look forward to extending it for suburban, rural, and even wild areas. Working with the relatively sparse network coverage and different application requirements could exhibit new challenges, but all these efforts will ultimately make our work beneficial to everyone everywhere.

Bibliography

[1] D. P. Abreu, K. Velasquez, M. Curado, and E. Monteiro. A resilient internet of things architecture for smart cities. *Annals of Telecommunications*, pages 1–12, 2016.

[2] M. AbuJayyab, S. A. Ahdab, M. H. Taji, Z. A. Hamdani, and F. Aloul. PolluMap: A Pollution Mapper for Cities. In *Innovations in Information Technology*, Nov. 2006.

[3] N. S. Alhassoun, M. Y. S. Uddin, and N. Venkatasubramanian. Safer: An iot-based perpetual safe community awareness and alerting network. In *IGSC*, pages 1–8. IEEE, 2017.

[4] N. S. Alhassoun, M. Y. S. Uddin, and N. Venkatasubramanian. Context-aware energy optimization for perpetual iot-based safe communities. *Sustainable Computing: Informatics and Systems*, 2019.

[5] F. Alt, A. Shirazi, A. Schmidt, U. Kramer, and Z. Nawaz. Location-based crowdsourcing: Extending crowdsourcing to the real world. In *NordiCHI*, pages 13–22, 2010.

[6] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Comput. Netw.*, 54(15), Oct. 2010.

[7] L. Balzano and R. Nowak. Blind calibration of sensor networks. In *ACM IPSN*, 2007.

[8] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker's guide to successful wireless sensor network deployment. In *ACM SenSys*, 2008.

[9] R. Beelen, O. Raaschou-Nielsen, M. Stafoggia, Z. J. Andersen, G. Weinmayr, B. Hoffmann, K. Wolf, E. Samoli, P. Fischer, M. Nieuwenhuijsen, et al. Effects of long-term exposure to air pollution on natural-cause mortality: an analysis of 22 european cohorts within the multicentre escape project. *The Lancet*, 2014.

[10] T. Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3), 2006.

[11] K. Benson, C. Fracchia, G. Wang, Q. Zhu, S. Almomen, J. Cohn, L. D'arcy, D. Hoffman, M. Makai, J. Stamatakis, and N. Venkatasubramanian. SCALE: Safe community awareness and alerting leveraging the internet of things. *IEEE Communications Magazine*, 53(12):27–34, Dec. 2015.

[12] K. E. Benson, Q. Han, K. Kim, P. Nguyen, and N. Venkatasubramanian. Resilient overlays for iot-based community infrastructure communications. In *IoTDI*, pages 152–163. IEEE, 2016.

[13] K. E. Benson and N. Venkatasubramanian. Improving sensor data delivery during disaster scenarios with resilient overlay networks. In *PERCOM Workshops*, pages 547–552. IEEE, 2013.

[14] K. E. Benson, G. Wang, N. Venkatasubramanian, and Y.-J. Kim. Ride: A resilient iot data exchange middleware leveraging sdn and edge cloud resources. In *IoTDI*, pages 72–83. IEEE, 2018.

[15] M. M. Bin Tariq, M. Ammar, and E. Zegura. Message Ferry Route Design for Sparse Ad Hoc Networks with Mobile Nodes. In *MobiHoc*, pages 37–48, New York, NY, USA, 2006. ACM.

[16] T. Black, V. Mak, P. Pathirana, and S. Nahavandi. Using Autonomous Mobile Agents for Efficient Data Collection in Sensor Networks. In *WAC*, pages 1–6, July 2006.

[17] K. D. Boese. *Cost versus distance in the traveling salesman problem*. UCLA Computer Science Department Los Angeles, 1995.

[18] A. Boubrima, W. Bechkit, and H. Rivano. Optimal deployment of dense wsn for error bounded air pollution mapping. In *DCOSS*, 2016.

[19] A. Capponi, C. Fiandrino, C. Franck, U. Sorger, D. Kliazovich, and P. Bouvry. Assessing performance of internet of things-based mobile crowdsensing systems for sensing as a service applications in smart cities. In *CloudCom*, 2016.

[20] Y. Chen, H. Hong, S. Yao, A. Khunvaranont, and C. Hsu. Gamifying mobile applications for smartphone augmented infrastructure sensing. In *NetGames*, pages 12:1–12:16, 2017.

[21] D. Christin, C. Rosskopf, M. Hollick, L. Martucci, and S. Kanhere. IncogniSense: An anonymity-preserving reputation framework for participatory sensing applications. In *PerCom*, pages 135–143, Mar. 2012.

[22] R. W. Clayton, T. Heaton, M. Chandy, A. Krause, M. Kohler, J. Bunn, R. Guy, M. Olson, M. Faulkner, M. Cheng, and others. Community seismic network. *Annals of Geophysics*, 54(6), 2012.

[23] G. Denker, N. Dutt, S. Mehrotra, M.-O. Stehr, C. Talcott, and N. Venkatasubramanian. Resilient dependable cyber-physical systems: a middleware perspective. *Journal of Internet Services and Applications*, 3(1):41–49, 2012.

[24] S. Devarakonda, P. Sevusu, H. Liu, R. Liu, L. Iftode, and B. Nath. Real-time Air Quality Monitoring Through Mobile Sensing in Metropolitan Areas. In *SIGKDD Workshop on Urban Computing*, UrbComp, pages 15:1–15:8, 2013.

[25] N. Do, Y. Zhao, C.-H. Hsu, and N. Venkatasubramanian. Crowdsourced mobile data transfer with delay bound. *ACM Transactions on Internet Technology (TOIT)*, 16(4):28, 2016.

[26] W. Dong, G. Guan, Y. Chen, K. Guo, and Y. Gao. Mosaic: Towards city scale sensing with mobile sensor networks. In *ICPADS*, pages 29–36. IEEE, 2015.

[27] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. The BikeNet Mobile Sensing System for Cyclist Experience Mapping. In *SenSys*, pages 87–101, New York, NY, USA, 2007. ACM.

[28] Z. Feng, Y. Zhu, Q. Zhang, L. Ni, and A. Vasilakos. TRAC: Truthful auction for location-aware collaborative sensing in mobile crowdsourcing. In *INFOCOM*, 2014.

[29] K. Fu, W. Ren, and W. Dong. Multihop calibration for mobile sensing: K-hop calibratability and reference sensor deployment. In *INFOCOM*, 2017.

[30] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11), 2011.

[31] W. Gao, G. Cao, A. Iyengar, and M. Srivatsa. Supporting cooperative caching in disruption tolerant networks. In *ICDCS*, pages 151–161. IEEE, 2011.

[32] Y. Gao, W. Dong, K. Guo, X. Liu, Y. Chen, X. Liu, J. Bu, and C. Chen. Mosaic: A low-cost mobile sensing system for urban air quality monitoring. In *INFOCOM*, pages 1–9. IEEE, 2016.

[33] D. Gavalas, I. E. Venetis, G. Pantziou, and C. Konstantopoulos. An iterated local search approach for multiple itinerary planning in mobile agent-based sensor fusion. In *2015 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, pages 1–7. IEEE, 2015.

[34] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[35] L. Guo and Q. Han. Reliable data collection from mobile users with high data rates in wireless sensor networks. In *WoWMoM*, pages 1–6, June 2012.

[36] J. Gupta, , and S. Lal. Comparison of some approximate algorithms proposed for traveling salesmen and graph partitioning problems. In *ICIoTCT*, 2018.

[37] S. Hachem, V. Mallet, V. Raphaël, P.-G. Raverdy, A. Pathak, V. Issarny, and R. Bhatia. Monitoring Noise Pollution Using The Urban Civics Middleware. In *IEEE BigDataService*, 2015.

[38] S. Hachem, A. Pathak, and V. Issarny. Probabilistic registration for large-scale mobile participatory sensing. In *PerCom*, pages 132–140, 2013.

[39] Q. Han, R. Eguchi, S. Mehrotra, and N. Venkatasubramanian. Enabling state estimation for fault identification in water distribution systems under large disasters. In *SRDS*, pages 161–170. IEEE, 2018.

[40] Q. Han, P. Nguyen, R. T. Eguchi, K.-L. Hsu, and N. Venkatasubramanian. Toward an integrated approach to localizing failures in community water networks. In *ICDCS*, pages 1250–1260. IEEE, 2017.

[41] Y. Han, Y. Zhu, and J. Yu. Utility-maximizing data collection in crowd sensing: An optimal scheduling approach. In *SECON*, pages 345–353, 2015.

[42] D. Hasenfratz, O. Saukh, and L. Thiele. On-the-fly calibration of low-cost gas sensors. *Wireless Sensor Networks*, 2012.

[43] D. Hasenfratz, O. Saukh, C. Walser, C. Hueglin, M. Fierz, and L. Thiele. Pushing the spatio-temporal resolution limit of urban air pollution maps. In *PerCom*, pages 69–77, 2014.

[44] H. Hong, P. Tsai, A. Cheng, M. Uddin, N. Venkatasubramanian, and C. Hsu. Supporting internet-of-things analytics in a fog computing platform. In *CloudCom*, 2017.

[45] H. Huang and A. V. Savkin. Path planning algorithms for a mobile robot collecting data in a wireless sensor network deployed in a region with obstacles. In *2016 35th Chinese Control Conference (CCC)*, pages 8464–8467, July 2016.

[46] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. CarTel: A Distributed Mobile Sensor Computing System. In *SenSys*, pages 125–138, New York, NY, USA, 2006. ACM.

[47] V. Issarny, V. Mallet, K. Nguyen, P.-G. Raverdy, F. Rebhi, and R. Ventura. Dos and Don'ts in Mobile Phone Sensing Middleware: Learning from a Large-Scale Experiment. In *ACM/IFIP/USENIX Middleware*, Dec. 2016.

[48] G. Jain, S. Babu, R. Raj, K. Benson, B. Manoj, and N. Venkatasubramanian. On disaster information gathering in a complex shanty town terrain. In *GHTC-SAS*, pages 147–153, Sept. 2014.

[49] R. Jalali, K. El-Khatib, and C. McGregor. Smart city architecture for community level services through the internet of things. In *ICIN*, pages 108–113, Feb. 2015.

[50] S. Kanhere. Participatory sensing: Crowdsourcing data from mobile smartphones in urban spaces. In *MDM*, pages 3–6, 2011.

[51] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE simulator for DTN protocol evaluation. In *ICST*, page 55, 2019.

[52] A. Khan, S. K. A. Imon, and S. K. Das. A novel localization and coverage framework for real-time participatory urban monitoring. *Pervasive and Mobile Computing*, 23:122–138, 2015.

[53] D. Kim, R. N. Uma, B. H. Abay, W. Wu, W. Wang, and A. O. Tokuta. Minimum latency multiple data mule trajectory planning in wireless sensor networks. *Mobile Computing, IEEE Transactions on*, 13(4):838–851, 2014.

[54] M. D. Kohler, T. H. Heaton, and M.-H. Cheng. The Community Seismic Network and Quake-Catcher Network: Enabling structural health monitoring through instrumentation by community participants. In *SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring*, pages 86923X–86923X. International Society for Optics and Photonics, 2013.

[55] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1), 2010.

[56] N. Künzli, R. Kaiser, S. Medina, M. Studnicka, O. Chanel, P. Filliger, M. Herry, F. Horak, V. Puybonnieux-Texier, P. Quénel, et al. Public-health impact of outdoor and traffic-related air pollution: a european assessment. *The Lancet*, 2000.

[57] N. D. Lane, Y. Chon, L. Zhou, Y. Zhang, F. Li, D. Kim, G. Ding, F. Zhao, and H. Cha. Piggyback CrowdSensing (PCS): Energy Efficient Crowdsourcing of Mobile Sensor Data by Exploiting Smartphone App Opportunities. In *SenSys*, pages 7:1–7:14, New York, NY, USA, 2013. ACM.

[58] S. Li, J. Chen, H. Yu, Y. Zhang, D. Raychaudhuri, R. Ravindran, H. Gao, L. Dong, G. Wang, and H. Liu. MF-IoT: A MobilityFirst-Based Internet of Things Architecture with Global Reach-Ability and Communication Diversity. In *IoTDI*, pages 129–140, Apr. 2016.

[59] C. Liao, T. Hou, T. Lin, Y. Cheng, A. Erbad, C. Hsu, and N. Venkatasubramania. SAIS: Smartphone augmented infrastructure sensing for public safety and sustainability in smart cities. In *EMASC*, pages 3–8, 2014.

[60] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi. Implementing Software on Resource-constrained Mobile Sensors: Experiences with Impala and ZebraNet. In *MobiSys*, pages 256–269, New York, NY, USA, 2004. ACM.

[61] Z. Lu, X. Sun, and T. La Porta. Cooperative data offloading in opportunistic mobile networks. In *INFOCOM*, pages 1–9. IEEE, 2016.

[62] K. L. Lueth. State of the IoT 2018: Number of IoT devices now at 7B – market accelerating. `https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/`. Online.

[63] J. Ma, N. Lu, and H. Zhang. Pso-based proactive routing in delay tolerant network. In *CCT*, pages 1–4. IET, 2014.

[64] A. Marjovi, A. Arfire, and A. Martinoli. High Resolution Air Pollution Maps in Urban Environments Using Mobile Sensor Networks. In *DCOSS*, 2015.

[65] S. Mehrotra, A. Kobsa, N. Venkatasubramanian, and S. R. Rajagopalan. Tippers: A privacy cognizant iot environment. In *IEEE PerCom Workshops*. IEEE, 2016.

[66] E. Miluzzo, N. D. Lane, A. T. Campbell, and R. Olfati-Saber. Calibree: A self-calibration system for mobile sensor networks. In *DCOSS*. Springer, 2008.

[67] A. Monfared, M. Ammar, E. Zegura, D. Doria, and D. Bruno. Computational ferrying: Challenges in deploying a Mobile High Performance Computer. In *WoWMoM*, pages 1–6, June 2015.

[68] M. J. Neely. Stochastic network optimization with application to communication and queueing systems. *Synthesis Lectures on Communication Networks*, 3(1):1–211, 2010.

[69] S. Nesamony, M. K. Vairamuthu, and M. E. Orlowska. On optimal route of a calibrating mobile sink in a wireless sensor network. In *IEEE INSS*, 2007.

[70] K. Ni, N. Ramanathan, M. Nabil-Hajj, and al. Sensor network data fault types. *ACM Transactions on Sensor Networks*, 5(3), 2009.

[71] AirUCI: Atmospheric integrated research at university of california, irvine. `http://airuci.uci.edu/`. Online.

[72] Array of things. `https://medium.com/array-of-things`. Online.

[73] Circuits framework. `http://circuitsframework.com/`. Online.

[74] Global City Teams Challenge (GCTC) program. `https://pages.nist.gov/GCTC/`. Online.

[75] GLPK (GNU Linear Programming Kit). `https://www.gnu.org/software/glpk/`. Online.

[76] GraphHoper. `https://www.graphhopper.com/`. Online.

[77] Gurobi. `http://www.gurobi.com/`. Online.

[78] Indoor mapping. `https://wiki.openstreetmap.org/wiki/Indoor_Mapping`. Online.

[79] Infrastructure canada - smart cities challenge. `https://www.infrastructure.gc.ca/cities-villes/index-eng.html`. Online.

[80] JOSM. `https://josm.openstreetmap.de/`. Online.

[81] MQTT. `http://mqtt.org/`. Online.

[82] OpenStreetMap. `https://www.openstreetmap.org/`. Online.

[83] QualNet. `http://qualnet.com/`. Online.

[84] SheevaPlug. `https://en.wikipedia.org/wiki/SheevaPlug`. Online.

[85] SmartAmerica Challenge. https://smartamerica.org/. Online.

[86] SoundCity: a mobile application for understanding your exposure to noise pollution. https://www.inria.fr/en/centre/paris/news/launch-of-soundcity-mobile-application. Online.

[87] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Sensing as a service model for smart cities supported by Internet of Things. *Transactions on Emerging Telecommunications Technologies*, 25(1):81–93, Jan. 2014.

[88] A. Petz, J. Enderle, and C. Julien. A Framework for Evaluating DTN Mobility Models. In *Simutools*, pages 94:1–94:8, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[89] A. Picchi. Verizon throttled firefighters' "unlimited" data during california fires, lawsuit claims. https://www.cbsnews.com/news/verizon-throttled-firefighters-unlimited-data-during-california-fires-lawsuit/, 2018. Online.

[90] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian. A software defined networking architecture for the internet-of-things. In *NOMS*, pages 1–9. IEEE, 2014.

[91] Z. Qin, L. Iannario, C. Giannelli, P. Bellavista, G. Denker, and N. Venkatasubramanian. MINA: A reflective middleware for managing dynamic multinetwork environments. In *NOMS*, pages 1–4, May 2014.

[92] C. Raffelsberger and H. Hellwagner. A hybrid manet-dtn routing scheme for emergency response scenarios. In *PerCom Workshops*, pages 505–510. IEEE, 2013.

[93] M. Rahman, H.-J. Hong, A. Rahman, P.-H. Tsai, A. Afrin, M. Y. S. Uddin, N. Venkatasubramanian, and C.-H. Hsu. Adaptive sensing using internet-of-things with constrained communications. In *Proceedings of the 16th Workshop on Adaptive and Reflective Middleware*. ACM, 2017.

[94] M. Rahman, A. Rahman, H.-J. Hong, L.-W. Pan, M. Y. S. Uddin, N. Venkatasubramanian, and C.-H. Hsu. An adaptive IoT platform on budgeted 3g data plans. *Journal of Systems Architecture*, 2018.

[95] R. Raj, S. Babu, K. Benson, G. Jain, B. S. Manoj, and N. Venkatasubramanian. Efficient Path Rescheduling of Heterogeneous Mobile Data Collectors for Dynamic Events in Shanty Town Emergency Response. In *GLOBECOM*, pages 1–7. IEEE, 2015.

[96] N. Ramanathan, L. Balzano, M. Burt, D. Estrin, T. Harmon, C. Harvey, J. Jay, E. Kohler, S. Rothenberg, and M. Srivastava. Rapid deployment with confidence: Calibration and fault detection in environmental sensor networks. *Center for Embedded Network Sensing*, 2006.

[97] D. Raychaudhuri, K. Nagaraja, and A. Venkataramani. Mobilityfirst: a robust and trustworthy mobility-centric architecture for the future internet. *ACM SIGMOBILE Mobile Computing and Communications Review*, 16(3):2–13, 2012.

[98] S. Romeo. 5 key insights from 350 smart city IoT projects. `https://iot-analytics.com/5-key-insights-from-350-smart-city-iot-projects/`. Online.

[99] F. Sailhan, V. Issarny, and O. T. Nascimento. Opportunistic multiparty calibration for robust participatory sensing. In *IEEE MASS*, 2017.

[100] S. Sarma, N. Venkatasubramanian, and N. Dutt. Sense-making from distributed and mobile sensing data: A middleware perspective. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[101] O. Saukh, D. Hasenfratz, and L. Thiele. Reducing multi-hop calibration errors in large-scale mobile sensor networks. In *ACM IPSN*, 2015.

[102] L. Scrucca. GA: A package for genetic algorithms in R. *Journal of Statistical Software*, 53(4):1–37, 2013.

[103] M. Sedighpour, M. Yousefikhoshbakht, and N. Mahmoodi Darani. An effective genetic algorithm for solving the multiple traveling salesman problem. *Journal of Optimization in Industrial Engineering*, 2012.

[104] J. A. Stankovic. Research directions for the internet of things. *IEEE Internet of Things Journal*, 1(1), 2014.

[105] M. S. Stankovic, S. S. Stankovic, and K. H. Johansson. Asynchronous distributed blind calibration of sensor networks under noisy measurements. *IEEE Transactions on Control of Network Systems*, 2016.

[106] D. Stone. Calibration and linear regression analysis: A self-guided tutorial.

[107] W. Sun, Q. Li, and C.-K. Tham. Wireless deployed and participatory sensing system for environmental monitoring. In *SECON*, pages 158–160. IEEE, 2014.

[108] R. T. Burnett, J. Brook, T. Dann, C. Delocla, O. Philips, S. Cakmak, R. Vincent, M. S. Goldberg, and D. Krewski. Association between particulate-and gas-phase components of urban air pollution and daily mortality in eight canadian cities. *Inhalation toxicology*, 2000.

[109] M. Talasila, R. Curtmola, and C. Borcea. Alien vs. mobile user game: Fast and efficient area coverage in crowdsensing. In *MobiCASE*, pages 65–74, 2014.

[110] M. Talasila, R. Curtmola, and C. Borcea. Crowdsensing in the wild with aliens and micropayments. *IEEE Pervasive Computing*, 15, Jan 2016.

[111] R. Tan, G. Xing, X. Liu, J. Yao, and Z. Yan. Adaptive calibration for fusion-based wireless sensor networks. In *INFOCOM*, 2010.

[112] J. Thelen, D. Goense, and K. Langendoen. Radio wave propagation in a patatoe field. In *1st workshop on wireless network measurement*, 2005.

[113] S. Toumpis and L. Tassiulas. Optimal deployment of large wireless sensor networks. *IEEE Transactions on Information Theory*, 52(7):2935–2953, 2006.

[114] D. Uckelmann, M. Harrison, and F. Michahelles. An architectural approach towards the future internet of things. In *Architecting the internet of things*, pages 1–24. Springer, 2011.

[115] M. Y. S. Uddin, A. Nelson, K. Benson, G. Wang, Q. Zhu, Q. Han, N. Alhassoun, P. Chakravarthi, J. Stamatakis, D. Hoffman, L. Darcy, and N. Venkatasubramanian. The Scale2 Multi-Network Architecture for IoT-Based Resilient Communities. In *SMARTCOMP*, pages 1–8, 2016.

[116] M. Y. S. Uddin, H. Wang, F. Saremi, G.-J. Qi, T. Abdelzaher, and T. Huang. Photonet: a similarity-aware picture delivery service for situation awareness. In *RTSS*, pages 317–326. IEEE, 2011.

[117] R. Vaisenberg, S. Ji, B. Hore, S. Mehrotra, and N. Venkatasubramanian. Exploiting semantics for sensor re-calibration in event detection systems. In *Proc. of SPIE*, volume 6818, 2008.

[118] A. Venkataramani, J. F. Kurose, D. Raychaudhuri, K. Nagaraja, M. Mao, and S. Banerjee. Mobilityfirst: a mobility-centric and trustworthy internet architecture. *ACM SIGCOMM Computer Communication Review*, 44(3):74–80, 2014.

[119] R. Ventura, V. Mallet, and V. Issarny. Assimilation of mobile phone measurements for noise mapping of a neighborhood. *Journal of the Acoustical Society of America*, 144(3), 2018.

[120] R. Ventura, V. Mallet, V. Issarny, P.-G. Raverdy, and F. Rebhi. Estimation of urban noise with the assimilation of observations crowdsensed by the mobile application ambiciti. In *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*, volume 255, pages 5444–5451. Institute of Noise Control Engineering, 2017.

[121] R. Ventura, V. Mallet, V. Issarny, P.-G. Raverdy, and F. Rebhi. Evaluation and calibration of mobile phones for noise monitoring application. *Journal of the Acoustical Society of America*, 142(5), 2017.

[122] O. Vermesan and P. Friess, editors. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. Publishers Series in Communication. River, 2013.

[123] T. Wang and C. P. Low. The general message ferry route (MFR$*$) problem and the An-Improved-Route (AIR) scheme. *Computer Networks*, 56(4):1442–1457, 2012.

[124] Y. Wang, A. Yang, X. Chen, P. Wang, Y. Wang, and H. Yang. A deep learning approach for blind drift calibration of sensor networks. *IEEE Sensors Journal*, 2017.

[125] R. Wohlers, N. Trigoni, R. Zhang, and S. Ellwood. Twinroute: Energy-efficient data collection in fixed sensor networks with mobile sinks. In *MDM*, pages 192–201. IEEE, 2009.

[126] S. Y. Wu and J. S. Liu. Evolutionary path planning of a data mule in wireless sensor network by using shortcuts. In *CEC*, pages 2708–2715, July 2014.

[127] B. Xing, S. Mehrotra, and N. Venkatasubramanian. Radcast: Enabling reliability guarantees for content dissemination in ad hoc networks. In *INFOCOM*, pages 1998–2006. IEEE, 2009.

[128] H. Xiong, D. Zhang, G. Chen, L. Wang, and V. Gauthier. CrowdTasker: Maximizing coverage quality in Piggyback Crowdsensing under budget constraint. In *PerCom*, pages 55–62, Mar. 2015.

[129] X. Xu, J. Gao, J. Gao, and Y. Chen. Air pollution and daily mortality in residential areas of beijing, china. *Archives of Environmental Health: An International Journal*, 49(4):216–222, 1994.

[130] M. Yuen, I. King, and K. Leung. A survey of crowdsourcing systems. In *PASSAT and SocialCom*, pages 766–773, 2011.

[131] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of Things for Smart Cities. *IEEE Internet of Things Journal*, 1(1):22–32, Feb. 2014.

[132] J. Zhang, Y. Sun, Z. Liu, D. Ji, B. Hu, Q. Liu, and Y. Wang. Characterization of submicron aerosols during a month of serious pollution in beijing, 2013. *Atmospheric Chemistry and Physics*, 14(6):2887–2903, 2014.

[133] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware Design Experiences in ZebraNet. In *SenSys*, pages 227–238, New York, NY, USA, 2004. ACM.

[134] M. Zhao, Y. Yang, and C. Wang. Mobile Data Gathering with Load Balanced Clustering and Dual Data Uploading in Wireless Sensor Networks. *IEEE Transactions on Mobile Computing*, 14(4):770–785, Apr. 2015.

[135] W. Zhao, M. Ammar, and E. Zegura. A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks. In *MobiHoc*, pages 187–198, New York, NY, USA, 2004. ACM.

[136] Y. Zhao, N. Do, S.-T. Wang, C.-H. Hsu, and N. Venkatasubramanian. O2SM: Enabling efficient offline access to online social media and social networks. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 445–465. Springer, 2013.

[137] Y. Zhao, C.-C. Liao, T.-Y. Lin, J. Yin, N. Do, C.-H. Hsu, and N. Venkatasubramanian. Smartsource: A mobile q&a middleware powered by crowdsourcing. In *MDM*, volume 1, pages 145–156. IEEE, 2015.

[138] Y. Zheng, T. Liu, Y. Wang, Y. Zhu, Y. Liu, and E. Chang. Diagnosing New York City's Noises with Ubiquitous Data. In *UbiComp*, pages 715–725, 2014.

[139] C. Zhu, C. Zheng, L. Shu, and G. Han. A survey on coverage and connectivity issues in wireless sensor networks. *Journal of Network and Computer Applications*, 35(2):619–632, 2012.

[140] Q. Zhu, M. Y. S. Uddin, Z. Qin, and N. Venkatasubramanian. Upload planning for mobile data collection in smart community internet-of-things deployments. In *SMARTCOMP*, pages 1–8, 2016.

[141] Q. Zhu, M. Y. S. Uddin, Z. Qin, and N. Venkatasubramanian. Data collection and upload under dynamicity in smart community internet-of-things deployments. *Pervasive and Mobile Computing*, 42:166–186, 2017.

[142] Q. Zhu, M. Y. S. Uddin, N. Venkatasubramanian, and C.-H. Hsu. Spatiotemporal scheduling for crowd augmented urban sensing. In *INFOCOM*, pages 1997–2005. IEEE, 2018.

[143] Q. Zhu, M. Y. S. Uddin, N. Venkatasubramanian, C.-H. Hsu, and H.-J. Hong. Enhancing reliability of community internet-of-things deployments with mobility. In *INFOCOM Workshops*, pages 1–2. IEEE, 2018.

[144] R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, and K. Sycara. Distributed constraint optimization for teams of mobile sensing agents. *Autonomous Agents and Multi-Agent Systems*, 29(3):495–536, 2015.

Appendices

Supplementary material goes here.

A Data Upload Planning: Proof of Complexity

We prove the constrained optimization formulation of the upload planning problem, shown in Equation (5.3), is NP-hard, even with all information made available, complexity significantly reduced, and no dynamicity introduced as is assumed in the static planning phase. Such computational complexity is proven by showing that the 0-1 knapsack problem, which is known to be NP-complete, can be reduced to an (actually simple case of) upload planning problem, where (a) There is only one upload opportunity that represents the knapsack; (b) There are multiple data chunks representing items, such that the size and priority of data chunks are the weight and value of items; (c) An on-time upload indicates an item being placed in the knapsack.

Consider the following 0-1 knapsack problem: Given a set of N items, numbered $1, ..., N$,

each with a value $v_i > 0$ and a weight $w_i > 0$, along with a maximum weight capacity W,

$$\begin{aligned} &\max \sum_{i=1}^{N} v_i x_i \\ &\text{s.t.} \sum_{i=1}^{N} w_i x_i \leqslant W, \text{and } x_i \in \{0,1\} \end{aligned} \tag{A.1}$$

According to our definition of utility, $f(\Delta) \in [0,1]$, and f should be monotonically decreasing over $\Delta \in \mathbf{R}$. Therefore, according to the monotone convergence theorem,

$$\lim_{x \to +\infty} f(\Delta) = 0$$
$$\lim_{x \to -\infty} f(\Delta) = 1$$

Let

$$\Delta_+ = \min \{\Delta \geqslant 0 | f(\Delta) \leqslant \min\{v_i\}/\max\{v_i\}\}$$
$$\Delta_- = \max\{\Delta \leqslant 0 | 1 - f(\Delta) \leqslant \min\{v_i\}/\max\{v_i\}\}$$
$$\Delta_0 = \max\{\Delta_+, -\Delta_-, 1\}$$

In this way, we can reduce the original 0-1 knapsack problem as in Equation (A.1) to an upload planning problem with $M = 1$ (only one) upload opportunity and N data chunks, along with randomly chosen positive constant moving speed $v(x) = V, \forall x > 0, x \neq x_1 > 0$ and data collecting rate R_a.

Let

$$T_c = 0$$

$$\boldsymbol{u}_1 = (x_1, r_1), \text{ where } x_1 > 0, \text{ and } r_1 = 1/2\Delta_0$$

$$x(\boldsymbol{a}_i) = 0$$

$$s(\boldsymbol{a}_i) = w_i / \min\{w\}, \forall i = 1, ..., N$$

$$p(\boldsymbol{a}_i) = v_i / \max\{v\}, \forall i = 1, ..., N$$

If we set all $d(\boldsymbol{a}_i), \forall i = 1, ..., N$ to

$$d(\boldsymbol{a}_i) = d_0 = \sum_{k=1}^{N} \frac{w_k / R_a}{\min\{w\}} + \frac{x_1}{V} + \frac{W/r_1}{\min\{w\}} + \Delta_0$$

then only a subset of all data chunks with their total size being no greater than $W/\min\{w\}$ can be uploaded before their common deadline. According to our previous settings, any data chunk that is scheduled to be overdue cannot contribute a utility more than even the least important data chunk that is scheduled on-time. In this way, choosing from items to put into the knapsack is equivalent to choosing from the data chunks to upload at $\boldsymbol{u}_1$, i.e. $x_i = \boldsymbol{\Lambda}_{i,1}$.

Hence, the 0-1 knapsack problem in Equation (A.1) can be reduced to an upload planning problem formulated as in Equation (5.3). This means our simplified formulation of the upload planning problem is at least as hard as the 0-1 knapsack problem, which is known to be NP-complete. In other words, the upload planning problem is at least as hard as the hardest problems in NP. Therefore, the upload planning problem is NP-hard.

B Data Upload Planning: Estimation of Utility

As we mentioned earlier in Section 5.2.3, the planner needs to make an estimation of the overall utility, i.e. the EWOU U_e, when making a plan for the MDC before it departs. Therefore, we provide detailed steps below to show how $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ can be calculated by static planners with expectations/estimation on several data/opportunity parameters, so that EWOU U_e can be computed based on t_e.

The estimated delivery time $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ is the sum of three major components: the total moving time $t_m(\boldsymbol{a}_i, v_e)$, the total data collection time $t_a(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \lambda)$, and the total uploading time $t_u(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda, l)$, referring to the time spent on the corresponding activities before $\boldsymbol{a}_i$ is uploaded at $\lambda(\boldsymbol{a}_i)$.

The speed of MDC $v(x)$ has no definition at any data site or any upload opportunity. However, for its estimation $v_e(x)$, if we assign a finite positive value for it at those sites, then the moving time can be computed with

$$t_m(\boldsymbol{a}_i, v_e) = \int_0^{x(\boldsymbol{a}_i)} \frac{dx}{v_e(x)}$$

This integral, however, might be too heavy for planners. An easier way to estimate the moving time is to estimate the average moving speed from the beginning of path to location x as $\bar{v}_e|_0^x$, so the estimated moving time

$$t_m(\boldsymbol{a}_i, v_e) = \frac{x(\boldsymbol{a}_i)}{\bar{v}_e|_0^{x(\boldsymbol{a}_i)}} \tag{B.2}$$

The total data collection time is the time spent on all data sites that are located before

$\lambda(\boldsymbol{a}_i)$. Since $\mathbf{C}_{i,j} = 1$ iff $x(\boldsymbol{a}_i) \leqslant x(\boldsymbol{u}_j)$, we have

$$t_a(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \lambda) = \sum_{k=1}^{N} \mathbf{C}_{k,\lambda_i} \cdot \left(T_c + \frac{s_e(\boldsymbol{a}_k)}{R_a} \right) \tag{B.3}$$

where T_c is an estimation of the time to establish connection at an upload opportunity. Therefore, for a specific $\boldsymbol{a}_i$, it takes $O(N)$ time to compute $t_a(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \lambda)$.

The total uploading time $t_u(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda, l)$ spent before $\boldsymbol{a}_i$ is uploaded can be further divided into three parts: the uploading time spent before $\lambda(\boldsymbol{a}_i)$, denoted as $t_{u1}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda)$, the uploading time spent at $\lambda(\boldsymbol{a}_i)$ to upload other data chunks before $\boldsymbol{a}_i$ according to l, denoted as $t_{u2}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda, l)$, and the time to upload $\boldsymbol{a}_i$ itself, denoted as $t_{u3}(\boldsymbol{a}_i, \{\boldsymbol{u}\}, \lambda)$.

Since $\{\boldsymbol{u}_j\}$ is given in increasing order of $x(\boldsymbol{u}_j)$, given an M-by-M up triangular binary matrix $\mathbf{B}$, where $\mathbf{B}_{i,j} = 1$ iff $i < j$, the uploading time spent before $\lambda(\boldsymbol{a}_i)$ can be written as

$$t_{u1}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda) = \sum_{h=1}^{M} \mathbf{B}_{h,\lambda_i} \cdot \left(T_c \cdot \bigvee_{k=1}^{N} \mathbf{\Lambda}_{k,h} + \sum_{k=1}^{N} \mathbf{\Lambda}_{k,h} \cdot \frac{s_e(\boldsymbol{a}_k)}{r_e(\boldsymbol{u}_h)} \right) \tag{B.4}$$

whose computation takes $O(MN)$ time for a specific $\boldsymbol{a}_i$.

The uploading time spent at $\lambda(\boldsymbol{a}_i)$ to upload other data chunks before $\boldsymbol{a}_i$ is

$$t_{u2}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda, l) = T_c + \sum_{k=1}^{N} \mathbf{\Lambda}_{k,\lambda_i} \cdot l(\boldsymbol{a}_k, \boldsymbol{a}_i, \lambda) \cdot \frac{s_e(\boldsymbol{a}_k)}{r_e(\lambda(\boldsymbol{a}_i))} \tag{B.5}$$

whose computation takes $O(N)$ time.

The time to upload $\boldsymbol{a}_i$ itself is

$$t_{u3}(\boldsymbol{a}_i, \{\boldsymbol{u}\}, \lambda) = \frac{s_e(\boldsymbol{a}_i)}{r_e(\lambda(\boldsymbol{a}_i))} \tag{B.6}$$

Then the estimated delivery time $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ can be acquired from Equations (B.2-B.6) for any $\boldsymbol{a}_i$ in $O(MN)$ time, using

$$\begin{aligned} t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) = {} & t_m(\boldsymbol{a}_i, v_e) + t_a(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \lambda) \\ & + t_{u1}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda) + t_{u2}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda, l) \\ & + t_{u3}(\boldsymbol{a}_i, \{\boldsymbol{u}\}, \lambda) \end{aligned} \tag{B.7}$$

Finally, we are able to calculate the EWOU $U_e(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ out of the estimated delivery time $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)$ of all data chunks using Equation (5.2). Therefore, the overall time complexity to calculated the EWOU of a given static plan λ is $O(MN^2)$.