Delft University of Technology
Master's Thesis in Embedded Systems

SMoT: A Smartphone-Based Mobile Testbed for Human-Centric Wireless Networks

Platon Efstathiadis

embedded software

TU Delft Delft University of Technology

SMoT: A Smartphone-Based Mobile Testbed for Human-Centric Wireless Networks

Master's Thesis in Embedded Systems

Embedded Software Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Platon Efstathiadis
P.Efstathiadis@student.tudelft.nl

19th August 2015

Author
Platon Efstathiadis (P.Efstathiadis@student.tudelft.nl)
Title
SMoT: A Smartphone-Based Mobile Testbed for Human-Centric Wireless Networks
MSc presentation
August 27th 2015

Graduation Committee

Prof. Dr. K.G. Langendoen (chair)	Delft University of Technology
M.A. Zuniga Zamalloa, PhD	Delft University of Technology
Dr. Alessandro Bozzon	Delft University of Technology

Abstract

Recently, wireless sensor networks (WSNs) are becoming vital to a wide range of application domains, from precision agriculture and smart buildings to health systems and monitoring of humans, animals, crowds and robots. In particular, there is an increasing number of sensor devices that are worn by persons who interact with them on a daily basis. In these applications, networks are formed by the persons who wear the sensor devices and as a result the mobility comes from them. This results in new protocols and applications for this kind of networks and new challenges arise. Human factors are crucial aspects in this case and researchers should consider them in their designs. The protocols and applications developed, should be tested and verified before their final deployment. We argue that a testbed, exposing the human characteristics is needed.

In this thesis, we define the fundamental requirements that a testbed for experimentation with human-centric wireless networks must fulfil. We design and implement a fully functional testbed that allows researchers to run experiments in a realistic and controlled testing environment. We show that our testbed has the appropriate features and tools to make the running of the experiments easy and effortless. In addition, our testbed gives the ability to the researchers to observe how human diversity and variability (like body orientation or walking speed) affect their work. Furthermore, we evaluated the mechanisms of our testbed and present how they affect the execution of an experiment.

Preface

This thesis presents the work I have done in the Embedded Software Group during my studies for completing my Master of Science in Embedded Systems. My interest for wireless sensor networks started during the first year of my master degree. The applications, the particularities and the challenges that these systems have, fascinated me from the beginning and this is the reason I chose to do my graduation project in this field. During my thesis I expanded my knowledge and met several notable people who helped me in many ways.

I would like to thank my daily supervisor Marco Cattani for his support and encouragements. He helped me with his ideas, taught me the insights and the proper way to conduct research and our discussions were always very instructive. My thanks also go to Marco Zuniga who helped me construct my thoughts and choose the right topic for my graduation project. Many thanks to my colleagues Ioannis, Michal, Dimitris, Michalis and all the people of the group. Furthermore, I would like to thank Koen Langendoen, for hosting me at the Embedded Software group and Alessandro Bozzon, for being a member of my graduation committee. Finally, I would like to thank my friends and family for their continuous support during all these years.

Platon Efstathiadis

Delft, The Netherlands
19th August 2015

Contents

Chapter 1

Introduction

Wireless sensor networks (WSNs) consist of tiny devices with limited processing power and sensors that communicate wirelessly with each other. They have become valuable and appropriate systems for a diverse range of applications and sectors. Precision agriculture, building management and health care as well as environment and human sensing are some representative examples. During the last years, the characteristics of sensor networks made them an effective and suitable solution for applications that involve humans, like human-activity recognition and crowd monitoring.

Energy efficiency, power, compactness and cost-effectiveness are some properties that enable wireless sensor networks to be used in applications that involve humans in contrast to others. For instance, video based systems exist for monitoring a crowd. In these systems, additional hardware components and high implementation cost are two main disadvantages. Moreover, tracking people in huge areas is difficult due to the limitations in equipment and privacy concerns.

Protocols and applications for wireless sensor networks typically need to be evaluated and tested before their final deployment to real-world solutions. This procedure helps the researchers to understand the behaviour of their protocols under specific conditions and settings. It also makes debugging more efficient and gives the opportunity to fix potential bugs before the final implementation of the application or protocol.

Tools that can be used for testing and verification are testbeds and simulations. Although simulations provide a controlled testing environment, they fail to provide a sufficient level of realism. Assumptions are made about the radio propagation, traffic and topologies, which led to many weaknesses as stated in Egea-Lopez's work [24]. On the other hand, testbeds are able to reveal the real vagaries of wireless communication and expose protocols and applications to real-world wireless sensor network scenarios without the idealities of simulation environments. Moreover, testbeds enable researchers to design robust applications or protocols, and test them on real hardware.

We can classify testbeds in different ways, depending on the kind of the applications that they have to assay. Applications and protocols for WSNs expose diverse characteristics. For example, some protocols aim for WSNs where the nodes are static in contrast to others where the nodes are mobile. As a result, there are testbeds with static nodes like [31] [27] [14] [23], testbeds with mobile nodes [29] [30] and testbeds that consist of other sub-testbeds [19]. The kind of testbed that a researcher is going to choose to test his application and protocol depends on the kind of real wireless sensor network that will be implemented.

The work in this thesis is motivated by the fact that applications and protocols for human-centric wireless sensor networks (networks that are created by devices that humans wear or carry) need a suitable testbed in order to be tested and verified. A testbed is required where experimentation conditions are as close as possible to the typical operating conditions of the final deployed solutions. For instance, crowd proximity [28], density estimation [18], human-activity recognition [35] and monitoring of the social behaviour [17] are applications and protocols where the human factor has a significant role. Existing mobile testbeds fail to capture human-like effects, like how body interference and orientation affects node connectivity. Realism is also something really important. Human movement and variability (e.g. every person performs the same instruction differently) are two aspects that applications and protocols under experimentation should be exposed to.

The challenge of providing a functional testbed that will take into consideration human characteristics and provide sufficient realism is not intuitive. A possible solution could be to give the sensor nodes of the wireless sensor networks to people. However, this way is not always feasible and has a list of drawbacks. Coordination of the persons, debugging of the results and assignment of identifiers (IDs) are some problems that this solution struggles to solve. Among these problems collection of the logging data, mass reprogramming of the testbed nodes and exposing the testbed nodes to the human variability are some challenges that the thesis is trying to address. The following question must be answered: "How can we create an effective mobile testbed for human-centric WSNs and solve the above challenges?"

The wide usage of smartphones inspired the design and implementation of our testbed. Modern smartphones have become ubiquitous and supplanted the desktop PC as the dominant mode for accessing the Internet. They constitute a major part in the everyday life of people as they carry them everywhere and use them in a variety of situations. They are equipped with a powerful processor, connectivity capabilities and a wide range of sensors that can be used to deduce information about the context of a user and the environment. These properties make smartphones eligible and appropriate components for the testbed we would like to deploy. We believe that the combination of these powerful and pervasive devices with the sensor nodes of a wireless sensor network will allow us to create a mobile testbed for sensor

networks oriented to humans.

SMoT, which is an abbreviation for Smartphone-based Mobile Testbed, is the name of our testbed. It is a low-cost, easy to deploy, mobile testbed that targets applications and protocols for human-centric sensor networks. A representative example of these applications is the estimation of a crowd's density. Increasing the realism and including humans in the experiments is something crucial for testing application and protocols like the above. By doing that, researchers will be able to better understand the behaviour of their work, fix contingent bugs and test them in a controlled realistic environment.

The contributions of the work presented in this thesis are the following. First, we define a complete set of requirements that a testbed for human-centric WSNs should have. This set includes basic requirements that are common to the majority of testbeds, but also some additional requirements such as participant coordination, ground truth and participant's feedback collection. Second, we designed and implemented a fully functional testbed for this kind of networks. We explain the challenges that we encountered as well as the existing limitations. Finally, we present the results of evaluating the mechanisms of our testbed, which contribute to the execution of an experiment for human-centric WSNs.

The rest of the thesis is organized as follows. In Chapter 2, we describe the requirements that a testbed for human-centric WSNs applications and protocols should fulfil. Chapter 3 presents existing mobile testbeds and in Chapter 4 we present our design choices with our motives. Chapter 5 describes the implementation of *SMoT*, followed by the evaluation of our testing infrastructure in Chapter 6. Finally, the conclusions are drawn in Chapter 7

Chapter 2

Requirements

The usage of sensor networks for human-centric applications opens up new challenges that demand new capabilities and features from testbeds. Existing mobile testbeds do not present the characteristics that make them able to cope with these kinds of applications and protocols.

After studying many surveys [25] [26] [32] [33] for a wide variety of testbeds, we analysed and defined a set of requirements for a mobile testbed for human-centric WSNs. There are requirements that are common to existing testbeds and are necessary in order to have a fully functional testing infrastructure. However, each testbed targets different domains and applications. Each application has its own characteristics and peculiarities that must be captured by the testbed. Thus, a testbed for human-centric applications and protocols must provide features that capture the human variability in realistic conditions and provide a sufficient amount of realism. In this thesis, we define and present the set of requirements that a testbed for human-centric WSNs applications must fulfil.

Human factor — One of the goals of a testbed is to provide controlled experimental conditions with a sufficient amount of realism. In particular, a testbed for human-centric wireless sensor networks must be able to capture and expose to the applications and protocols the human variability and characteristics like moving speed and body orientation, which can significantly affect the results. A representative example can be found in Rensfelt's work [30], where the mean RSSI value measured from a sensor node carried by a person and a robot shows significant differences. Furthermore, for some applications interacting with humans is a vital requirement since it helps to collect the ground truth. However, by involving humans, privacy should be taken into consideration. Mechanisms are required to protect persons' data, control what information they share and grant their permission for taking part in the testbed.

Testbed Usability — From the researchers perspective, the testbed should provide the appropriate tools and services to easily access, setup and run an experiment. Remotely accessing the testbed is a basic functionality of any testbed that allows the researchers to access the testbed at any time and from any place without the need to visit the testbed's location. Providing an interface, which researchers can easily use, helps to set up and configure an experiment. Since the sensor nodes need to be programmed in order to run an experiment, pushing their firmware on the fly enables the massive deployment of an experiment with less time and effort from both the researcher and participants. Another important aspect for the usability of a testbed is the ability to synchronize the testbeds events with the experiment's events. To achieve this goal, being able to synchronize the experiment execution across several nodes is an essential characteristic, which allows the testbed to start the nodes in a simultaneous manner of a few seconds. Moreover, parsing the results and debugging the protocols and application can be two challenging procedures, especially when the logs are distributed and not synchronized. Therefore, timestamping the results and any other event makes these procedures more straightforward.

From the participants perspective, the testbed should not interfere with their work flow. It should interrupt the participants only when it is necessary for the experiment. Furthermore, the testbed nodes should be able to be deployed in a small amount of time and effort, which will also make the participation in the testbed and the redeployment of the infrastructures easier.

Mobility — In human-centric WSNs the sensor nodes are attached to persons that freely move around in a real-world environment. As a result, mobility is one of the main characteristics that our testbed should support and provide.

We can categorize mobility in two forms, *Passive* and *Active*. *Passive* refers to the mobility of a device embedded in an object or carried by a person, which cannot be controlled by the device itself. This kind of mobility can be either non-predictive or predictive. An example of non-predictive mobility is a human or an animal that can freely move through space. If an observer watches their movements, he will not be able to predict their next move. On the contrary, a bus that has a predefined and acquainted route to follow, is an example of predictive mobility. Despite the fact that we know the route of the bus, we are not able to control it. While predictive mobility offers larger possibilities in terms of applications, controlling and exploiting the results is a real challenge.

On the other hand, *Active* mobility refers to the object/person that is moving following real time or predefined navigation instructions. Robots usually provide this type of mobility. They can follow lines on the ground, specific coordinates or pre-defined traces in a very accurate and precise way.

However, when using robots localization and battery charging are two additional challenges.

For both humans and robots, handling mobility and the associated dynamics is thus a key requirement for mobile WSN testbeds. Therefore, mechanisms to control and exploit realistic human mobility during experimentation are necessary.

Cost — The deployment of a testbed includes some cost. The hardware, the equipment needed, the connections between devices (and the experimenter) and the maintenance are the main sources of expenses. Cost is an important aspect of a testbed, since it is one of the main factors that limits the testbed operation. The objective is to have a testbed with cost per node close to or less than the cheapest static testbed for WSNs in the literature, Indria [23]. By using off-the-shelf-hardware we can reduce the cost of the testbed and make also the deployment easier.

Scalability — Human-centric wireless sensor networks can consist of tens, hundreds and even thousands of nodes that can be deployed in both an indoor and an outdoor environment. Movement can range from the size of a building up to the scale of a city. Thus, supporting experiments on a wide range of scales is an important aspect that contributes toward the realism of human-centric WSNs. In order to make the experimentation with mobile sensor nodes at these scales possible, the way that nodes are added or removed from the testbed should be simple and fast. Testbed designers should consider that in their designs in order to deploy an easy to extent testing infrastructure.

Participant Coordination — For robot-based mobile testbeds coordinating the movements is simple. The researcher can create traces that the robots are able to follow precisely (given a precise positioning system) in different ways. On the other hand, in a testbed with people coordination is a vital requirement. However, coordinating people, can be a very challenging procedure. Imagine the case of a researcher giving directions to tens or hundreds of people who participate to his experiment. Some of them may not listen to what he said, leading to a number of consequences like delayed or wrong movements. To avoid this kind of situations a testbed should be able to guide the participants during an experiment.

Coordinating the participants individually, gives to the researcher the opportunity to have more control over the experiment. It makes the repetition of an experiment in similar conditions feasible and better than in the case where there is no guidance from the testbed. This also helps researchers to compare and understand the behaviour of their applications and protocols. By giving instructions, the testbed needs to have a way to capture the participant's reaction to those instructions. Thus, the testbed should

have a way to collect the feedback from the participants. In addition, the testbed should have mechanisms that allow the participants to interact with the testbed and produce events if an experiment demands it.

Testbed Lifetime — In a wireless mobile testbed the main power supply is a battery. A battery-powered device is able to operate for a specific duration depending on the capacity of the battery and the power consumption of the device. As a consequence, the testbed nodes are able to operate for a specific amount of time depending on their consumption. This places a boundary on the amount of time that an experiment can run. We believe that running an experiment for at least one hour per day is a logical low boundary. However, by using battery-powered devices, a charge mechanism is needed in order to keep the devices alive. Therefore the testbed should have a self-chargeable mechanism, which will allow the recharging of the batteries of a testbed node at least once per day.

According to our knowledge there is not a mobile testbed that fulfils all of the above mentioned requirements. As we will discuss in Chapter 3, existing mobile testbeds offer some of these features like mobility or self-recharging mechanisms but they are not suitable for testing human-centric sensor networks applications and protocols. They fail to address how the human factors affect the protocols and applications in a controlled and realistic environment.

Chapter 3

Related Work

In this chapter we analyse existing mobile testbeds for wireless sensor networks, which have some common features and fulfil partially the requirements described in Chapter 2. The majority of the existing testbeds are using robots to provide mobility. However, the use of robots implies some overheads. Necessity for the design of moving patterns, obstacle and collision avoidance mechanisms and movement restrictions are some examples. Moreover, the mobility patterns that robots will follow cannot be as random as human movement and cannot be compared to the real true mobility of humans.

Mobility is one of the main differences between existing mobile testbeds and *SMoT*. In our testbed the mobility comes from persons, the participants of the testbed. This way, *SMoT* can expose the applications and protocols to the human characteristics such as unpredictable walking speed, body orientation etc. In the following, we describe each individual mobile testbed and analyse the reason why they do not completely fulfil our application requirements.

MiNT-m [22] is a platform that supports a large number of experiments for mobile multi-hop wireless network protocols. In order to support mobility and to allow a flexible reconfiguration, each node is mounted on a centrally controlled mobile Roomba [10] robot. Robots are located in a room with a camera-based localization system in the ceiling for navigation purposes. It has a number of special features like real-time visualization of the testbed activity, topology reconfiguration, comprehensive control and monitoring of node's position. It also has an automatic battery recharging mechanism, like *SMoT* but provided in a different way, supplied by Roomba robots which makes it capable to operate 24/7.

However, the scalability of *MiNT-m* is very limited. The need for multiple charging stations and several cameras, accurately installed on the ceiling in order to provide positioning and localization of the nodes are two aspects that reduce its scalability. Furthermore, as we can see in Table 3.2, the cost

of the facility and nodes makes the deployment of the complete infrastructure expensive. In contrast, *SMoT* is able to scale from the size of a room to a city with low cost and without the need of complex mechanisms for keeping the testbed alive.

The *SCORPION* [15] testbed has as its main target heterogeneity. In particular, this mobile testbed consists of four different types of sensor nodes. Aerial and ground robots, nodes located in the buses of university campus and nodes inside a briefcase carried by persons are the nodes of this testbed. It combines mobility that comes from both robots and humans that are able to move around in an indoor and outdoor environment. Unfortunately, *SCORPION* does not provide any autonomous recharging mechanism, remote access or reprogramming capabilities like *SMoT*. It is also an expensive testbed with limited scalability. Nevertheless, it is suitable for testing and evaluating a wide range of mobile protocols because of all the types of mobility it provides.

Sensei-UU [30] uses robots, paired with a camera that follow pre-defined paths created with lines on the floor. The utilization of robots provides mobility precision to the facility that is difficult to achieved with human mobility. In addition, one of the main advantages is that it can be deployed easily in different places just by re-placing the guiding lines on the floor. The *Sensei-UU* does not have features like remote access, self-rechargeable mechanisms and bulk reprogramming of nodes because its scope is to be deployed only when it is needed and not to be a permanent testing infrastructure.

Model trains running on tracks are the main components of *TrainSense* [31]. The testbed provides restricted mobility through fixed tracks. The ability to track and localize the nodes as well as the reliable source of energy, which is able to be provided continuously through the rails, are two main advantages of this testbed. Moreover, it uses a portable device that can be paired with the trains in order to reprogram the nodes and download the results. The downside of this system is that the rails limit the scalability and flexibility of the testbed.

Different from the previous mobile testbeds, which were targeting WSNs and mainly using robots to provide mobility, *PHONELAB* [29] and *Pogo* [16] are large-scale testbeds that consist only of Android smartphones and focus on mobile phone sensing and smartphone research. That is, the researcher can request and collect data from the different sensors of the participant's phone. In *PHONELAB*, the researcher develops an Android application (experiment) locally, tags by using the *PHONELAB's* library the sensor data of the phone he is interested in and requests an upload of his application (experiment) to *PHONELAB*. The experiments (applications) are distributed to the testbed nodes (the participants smartphones), through the Play Store or over-the-air. The collection of the results is performed only when the phone is charging to avoid draining the battery. In *Pogo*, the researchers

	MiNT-m	SCORPION	Sensei-UU	TrainSense	PHONELAB	Pogo	SMoT
Mobility	robots	robots & humans	robots	robots	humans	humans	humans
Scalability	Low	Medium	Medium	Low	Medium	Medium	Medium
Participant Coordination	NA	NA	NA	NA	Yes	Yes	Yes
Autonomous Charging	Yes	No	No	Yes	Yes	Yes	Yes
Environment	Indoor	Both	Both	Indoor	Both	Both	Both
Remote Access	No	No	No	No	Yes	Yes	Yes
Re-programming	No	No	No	Yes	NA	NA	Yes

Table 3.1: Testbed comparison. Existing robot-based testbeds for WSNs are not able to fulfil all the requirements for a human-centric WSN since they lack the involvement of humans. On the other hand, the testbeds that involve humans fulfil the requirements, but their scope is not the WSNs. *SMoT* is able to fulfil all the requirements for a human-centric WSN while it combines sensor nodes and humans.

write their experiments in JavaScript and, by using the testbed's API, are able to run an experiment, access the smartphone's sensor data and collect the results. The testbed provides the necessary software, which runs in the smartphones of the participants and the PC of the researcher, in order to translates the API functions to Android calls. By doing so, it hides the communication between the researcher and participant, which helps the usability of the testbed.

Both testbeds inspired our design and choices. In our solution, the humans have a very significant role in the testbed too. They provide the mobility and the autonomous charging mechanism. The participants of *PHONELAB* and *Pogo* use the smartphones of the testbed as their regular phones. As a result, in order to be functional the participants charge their phones at least on a daily basis. Exploiting that fact the testbeds can operate 24/7. In addition *PHONELAB* and *Pogo* are able to coordinate the participants by using the smartphone capabilities. Furthermore, both testbeds are easily scalable and provide realistic human movements (their intrusiveness is minimal). The participant's privacy is handled in both *PHONELAB* and *Pogo* using sandbox techniques. An experiment can access only data for which the participant gave his permission at the beginning of the experiment.

Table 3.1 compares the testbed characteristics with the requirements mentioned in Chapter 2. In addition, an overview of the deployment cost of the testbeds is presented in Table 3.2. The table shows the per-node cost for each testbed and the cost of the infrastructure needed to manage the testbed. As we can observe, *SMoT* has the lowest cost of the mobile testbeds for WSNs while fulfilling all the requirements. The cost per testbed node in *SMoT* includes the price of the cheapest smartphone ($59.8) that supports our requirement (USB OTG capabilities), the price of the sensor node ($74.41) and the USB OTG cable ($0.79). However, if we use the same phone the cost per node of *PHONELAB* and *Pogo* can be less than *SMoT* since they require only the smartphone. In the case of *PHONELAB* we mention the cost of the smartphones used in the testbed as provided in the paper. The

Testbed Name	Cost/Node	Facility Cost	Number of Nodes	Testbed Cost
MiNT-m	$ 1665	$ 2100	12	$ 22080
SCORPION	Aerial Node: $ 464 Bus Node: $ 360 Briefcase Node: $ 282 Ground Nodes: $ 402	$ 0	Aerial Node: 8 Bus Node: 40 Briefcase Node: 20 Ground Nodes: 20	$ 31972
Sensei-UU	$ 696	$ 20	-	-
PhoneLab	$ 195	$ 0	288	$ 56160
Indria	$ 158 (average per node)		127	$ 20066
SMoT	$ 134	$ 0	100	$ 13433

Table 3.2: Cost per testbed. Note that the *Indria* [23] testbed is a static testbed and the reason we presented here is because it is the lowest costly static testbed.

papers of *Pogo* and *TrainSense* do not provide any details about the cost of each node. In addition, in the paper of *Sensei-UU* there is not any reference to the total number of nodes since it is not a permanent testbed.

Chapter 4

Design

The basic idea behind the design of our testbed is the use of smartphones as a testbed infrastructure. We attach the sensor nodes, which are the devices that run an experiment, to the smartphones, which act as the means of control and communication. We call the pair of a *smartphone* and a *sensor node*, a *testbed node* and each testbed node is carried by a person that participates in the testbed. This is not a limitation as a participant can carry more than one testbed node, but it depends on the experiment requirements defined by the researcher. The choice of smartphones gives us a number of advantages and enable us to fulfil the requirements, mentioned in Chapter 2, for a human-centric WSN testbed.

First, the fact that the participants of the testbed carry the sensor nodes gives the researchers the ability to exploit the natural mobility of humans. Since we design a testbed for testing applications and protocols for sensor devices that will be carried or worn by persons, providing realistic mobility is a vital requirement. The participants of the testbed are able to move around in a real-world indoor and outdoor environment without limiting the movements inside a particular area like the existing solutions with robots.

Second, by attaching the sensor nodes to the participants smartphones our testbed can capture and expose to the researchers the human characteristics and variability in a realistic environment. Moreover, since the participants of the testbed can use their own smartphone as part of the testbed node, our testbed is equipped with an autonomous self-rechargeable mechanism. The mechanism relies on the fact that the participants use their phone in many aspects of their life for making calls, sending messages or taking pictures. As a result, they have to charge it regularly (we assume at least once per day) in order to use it. Exploiting this circumstance our testbed can operate in 24/7 and be available at least once per day.

Third, the powerful capabilities, the wide range of sensors and the connectivity features, which modern smartphones have, makes them suitable choice for remotely accessing the testbed, run complex tasks like repro-

gramming the sensor nodes and interact with the participants. Using smartphone's screen, buttons, sensors, the testbed is able to coordinate the participants of the testbed and collect their feedback as well as the ground truth. In addition, the sensors of the smartphone can be utilized in order to provide feedback information to the testbed without demanding any actions from the participants. For example, instead of requiring from the participant of the testbed to press a button when he will move to a specific location, the GPS of the phone can detect that and log it in a file.

Finally, the design of our testbed allows easy deployment without spending a large amount of effort and money. Smartphones are common and cheap devices (depending though on the specifications).

Besides the testbed nodes, a central server is part of our testbed. In *SMoT* we call it the *testbed manager*. *SMoT* follows a master-slave approach similar to a large majority of testbeds like Pogo [16], PHONELAB [29] and SenseiUU [30], in which the master is the *testbed manager* and the slaves are the *testbed nodes*. The testbed manager provides an interface to the researchers in order to be able to interact with the testbed. The researchers can manage and control the testbed nodes, start and configure an experiment and collect the results.

Figure 4.1 presents an overview of the testbed components as well as the human entities involved in the system architecture, while Figure 4.2 shows the relations between them.

For clarity we present an example usage scenario of an experiment that can run in our testbed. The scenario demonstrates and makes the understanding of the testbed design easier. Imagine the case where a researcher wants to run an experiment for testing his protocol in the corridor of a building. The researcher has gathered a group of people and gave each of them a sensor node to hold, which communicates with the other sensor nodes and performs some computations. The persons move to a number of waypoints, which are specific locations in the corridor and it is the researcher's responsibility to set and manage these locations, when the researcher instructs them. The researcher is interested in the results of the protocol at each specific waypoint. After the end of the experiment, the researcher has to collect the results from the sensor nodes.

4.1 Testbed Node

In *SMoT*, a testbed node is a sensor node that is tethered together with a smartphone. The sensor node is the device that runs the experiment and communicates with the other sensor nodes. It is a device equipped with a microprocessor, a transceiver, a flash memory and a set of sensors. While usually a sensor node is powered by its own battery, in *SMoT* the battery of the smartphone provides the power to the sensor node, which has some

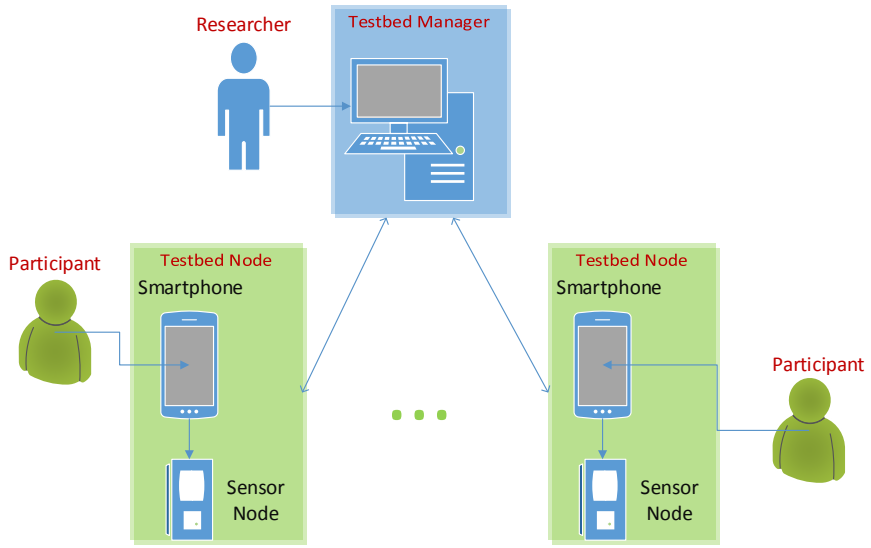


Figure 4.1: Overview of SMoT components.

advantages and disadvantages. It makes the testbed more usable since the deployment of, and the participation in the testbed becomes easier removing the need for periodically changing the sensor node's batteries. Nevertheless, powering the sensor node with the battery of the smartphone reduces the battery life of the testbed node because the same battery has to provide power to two devices, the smartphone and the sensor node. We discuss that in Section 6.3, where we present a detailed analysis of the power consumption of the testbed node.

Besides the power, the smartphone provides the main functionalities of the testbed node.

First, the client part of the software of the testbed, which runs in the smartphone, has the ability to communicate, program, reset the sensor node and collect the results after the end of an experiment. This allows on-the-fly reprogramming of the sensor nodes and reduces the effort and time needed for both the researcher and the participants. In our example, the researcher is able to run several experiments by commanding the smartphones to reprogram the sensor nodes with the new firmware. It is not necessary to gather all the sensor nodes and distribute them again to the participants.

Second, the smartphone establishes a connection with the testbed manager. Therefore, it can be remotely accessed, receive the necessary files of an experiment and upload the results to the testbed manager. Finally, the testbed software running in the smartphone of the participant is able to interact with the participant, collect his feedback and the ground truth and

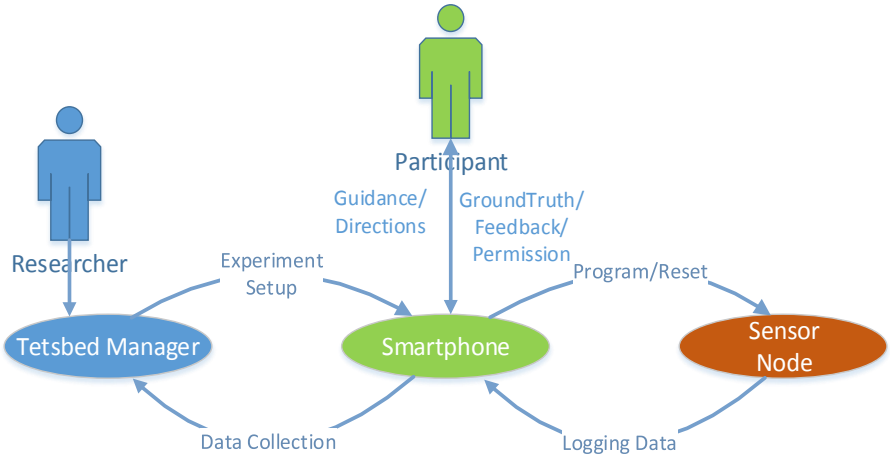


Figure 4.2: The testbed components with the supported functionalities between them.

guide him. In the usage scenario, which we presented earlier in this chapter, by using *SMoT* the researcher can easily guide the participants when and to which waypoint to move. The participants, in turn, are able to interact with the testbed and confirm their arrival to a particular waypoint. As we already mentioned, coordinating the participants and collecting their feedback is a challenging process, but assists the researcher in having better control of his experiment, which improves the consistency of the results. Mobile testbeds as *PHONELAB* [29] and *Pogo* [16] that consist of smartphones, provide similar mechanisms to communicate with the researchers and guide the participants, but they do not provide an interface to the sensor node since it is out of their scope.

4.2 Participants

The participants are the persons that carry a testbed node and take part in the experiments. They are able to interact with the testbed and the experiments by using the smartphone of the testbed node. They can give also their feedback to confirm a requested action from the researcher, either as part of the application or the scenario under experimentation. They can provide the ground truth for an experiment from the point of the application user. The smartphone records every interaction with the participant and any guidance message produced by the smartphone given a set of instructions from the researcher. This is an important feature that helps the researcher to parse and debug his results in a more understandable and smart way,

since he has a way to know the ground truth and what the events were during an experiment.

Participation in an experiment is an important aspect in our approach, since our testbed needs persons to operate. Therefore, we designed our testbed in a way that the participation is as easy as possible. Similar to *PHONELAB* [29] and *Pogo* [16], in *SMoT* the participants have to install and run the software once on their phones and they do not have to perform extra actions unless the experiment demands it. It is a non-intrusive approach in which the software runs silently in the background without disturbing the participants from their current flow. However, the software can also run in the foreground when it is necessary. For instance, when interaction with the participants is required. To simplify the deployment process, the attachment of the sensor node to the smartphone follows a plug-n-play manner. The participants just connect the sensor node to the phone, using a cheap USB-OTG cable. Moreover, we guarantee that we take into consideration the participant's privacy and acquire his permission to participate in an experiment by using the smartphone's permission dialogues when the participant installs the software of the testbed. Permission dialogues are a common way to deal with privacy issues in smartphone applications and it is a mechanism that every modern operating system is equipped with.

To attract participants to our testbed, there are several policies. Like in *Pogo* [16], we anticipate that research institutions will be able to provide the testbed nodes to their employees or students and deploy the testbed. They can motivate the participation in the testbed by giving study credits to the students or some other kind of rewards. There are also other options to gather participants like PHONELAB [29], where the persons that take part in the testbed get a discount on their mobile contracts.

4.3 Testbed Manager

Our testbed follows a master-slave approach where the master is the *testbed manager*. It is the component that provides an interface to the researchers and it can run on any personal computer, which makes it portable. The researchers can use it to have remote access to the testbed in order to keep track of the status of the nodes that are on line, delegate tasks, send commands and firmwares and collect the results.

The testbed manager exposes to the researcher various commands. The set of commands includes the display of the *registered* and *online* testbed nodes, the execution and termination of an experiment and the collection of the results. The state of a testbed node can either be *online* or *offline*. A testbed node is *online* if the smartphone has an active Internet connection and a sensor node attached to it, which makes it ready to run an experiment, otherwise it is *offline*. When a testbed node requests to be part of the

Type	Files	Parameters
plain	firmware	serial output, synchronous start, experiment duration
guided	firmware, instructions	serial output, synchronous start, experiment duration

Table 4.1: Type and parameters of an experiment.

testbed, it registers itself to the testbed manager, which has a database that stores the registered testbed nodes. Thus, a *registered testbed node* is a node that is part of the testbed, but it may be online or offline at a certain moment.

By using the testbed manager, researchers are able to start and terminate an experiment. The experiments in our testbed have a specific format, which is characterized by two aspects, their type and the set of parameters (see Table 4.1). The *plain* type, which consists only the firmware to upload on the sensor node and the *guided* type, which consist of the firmware and the instructions to coordinate the participants. With the latter, researchers are able to give directions to the users of the testbed like guide them to move to a specific point, or perform a particular action. For instance, in the waypoint scenario we mentioned at the beginning of this chapter, the instructions of the guided type of the experiment is a list with the number of the waypoints and the amount of time the participants should stay at each waypoint.

Independent of the type of the experiment, the parameter set is the same. The first parameter is about the logging data. The researcher can specify if he is interested in collecting the serial output of the sensor node (if there is any, depending on the firmware). Synchronous start of the experiment is the second configurable parameter. That is, our testbed is able to start the sensor nodes simultaneously with a difference in the starting time of a few milliseconds. This is useful for instance, in case where the researcher needs to add simultaneously a number of nodes and needs the devices to simultaneously perform an action.

The last parameter is about the duration of an experiment. Note that, the researcher should be careful with this value because it should not exceed some limits, which we compute and present in Section 6.3. These limits are introduced by the fact that the testbed nodes are battery powered resulting in bounds on the duration of an experiment related to the energy available in the battery of the smartphone.

Up to now, we discussed the configuration of an experiment, but experiments give back results too. These results, the logging data, have a particular structure in *SMoT* in order to be easy comprehensible and help the debugging procedure. They are composed by the output of the sensor nodes and some annotations like the participant's feedback. Every output and every event is timestamped by the smartphone. We use the operating system's network protocol for clock synchronization in order to provide a global time for our timestamping mechanism with millisecond accuracy.

By doing so, our system can produce understandable and easy to compare logging data.

Furthermore, logging data can be enriched by smartphone's sensor data such as GPS or access point names or participant's feedback in order to record what is happening during an experiment. An example of that is annotations showing that the participant has moved, in our usage scenario, to a particular waypoint. Note that this information was generated by the participant through pressing a button in the user interface of the *SMoT* application running on the smartphone. The smartphone also annotates the time that the participant should move to the next waypoint and notifies him to perform the movement.

An important aspect here is how experiments and information in general can be distributed between the *testbed manager* and the *testbed nodes*. The choice of the method used is vital since it will affect the duration and robustness of an experiment.

Broadly speaking two methods exist in the literature for information dissemination, *pull-based* and *push-based* systems. In the *pull-based* method the client or the user starts the communication process by requesting something from a server, which respectively responds to the request. A typical example of such a system is the Android Play Store [4] and iPhone App Store [1]. In both systems the user selects a number of applications to download and the system sends the applications to the user's phone. The choice of which application will be installed to the user's phone lies completely with the user. In contrast, *push-based* systems allow the servers (the researchers in our case), to send their experiments to the testbed nodes without requiring an action from the participant of the testbed. This can be automatic like in AnonySense [20], or manual like in Prism [21].

In *SMoT*, like in *Pogo* [16], we have chosen a *push-based* system for disseminating information between the testbed manager and the testbed nodes. With this method the testbed does not bother the participants with unnecessary messages and changes in their work flow. In addition, the researchers are able to run an experiment without waiting several minutes or even hours as in the case of *pull-based* systems for the participant to receive and start an experiment. In *SMoT*, the distribution of an experiment needs a few seconds. Although this communication system is suitable for our testbed, it limits the freedom of the participants in terms of not being able to select what and how many experiments will run on their phones. As a result, in order to ensure that participants have the full control of their privacy and freedom, we always acquire beforehand their permission to participate in the testbed.

Chapter 5

Implementation

This chapter describes our implementation of *SMoT*, which is written in Java (more than 5K lines of code). The server part runs on a desktop PC. The client part instead runs on Android smartphones. Android was a suitable choice for a number of reasons. First, at the time of writing this thesis Android had the biggest share of the market. Second, it supports the USB On-The-Go (OTG) mode [12] (in contrast with iOS that does not have software support). Third, the cost of an Android smartphone with USB OTG support can be very low. By utilizing the OTG mode of the USB protocol in our implementation we are able to provide a serial bidirectional connection between the smartphone and the sensor node. The smartphone, by acting as master, can have control of a sensor node. In contrast to the serial connection, a wireless connection could also provide a medium of communication with the sensor node by using for example the smartphone's Bluetooth connectivity. However, a wireless connection would make the testbed more complex and a number of important operations like the programming of the sensor nodes difficult to achieve or even infeasible.

In addition, by using a USB OTG connector the smartphone can power the sensor node. A USB OTG connector is a normal USB connector with extra modifications and when it is connected to the phone drains constantly its battery even when there is no device connected. Alternative solutions could be to power the sensor nodes directly from the battery of the smartphones (requires soldering) or to use the audio jack connection of the phone in order to enable and disable the OTG mode. Nevertheless, these solutions demand extra modifications to the smartphones, to the USB OTG connector, which make the setup of a testbed node difficult and reduce the usability of our solution.

The current implementation of *SMoT* supports only one type of sensor node, the G-Node G301 [11] from SOWNet technologies. It has a CC1101 radio chip, 8 KB RAM, 8 MB of external flash memory, 8 GPIO pins for external connectivity and an MSP430 microproccessor. It is also equipped

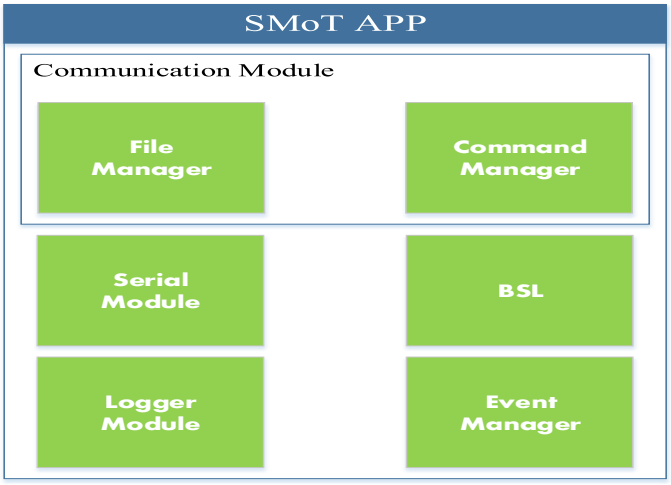


Figure 5.1: SMoT APP software architecture.

with a USB interface by using a USB controller from FTDI [2]. These capabilities make it suitable for wireless sensor network applications and deployments and a reasonable choice for our testbed. Although our implementation supports only one sensor node, adding a new device will not require a significant amount of effort. Only minor changes in specific software modules are needed because of the diversity in sensor node's hardware architecture. We have designed and implemented our software in a modular way in order to increase the extensibility of our testbed.

5.1 SMoT APP

The SMoT APP is the software part of the testbed that runs on the smartphone of a testbed node. It can run on Android 3.1 and higher because starting from that version, the software supports the USB OTG mode. The current version of the SMoT APP is approximately 4K lines of Java code. It is composed of six modules (see Figure 5.1) that interact between each other and provide the functionalities of a testbed node. The SMoT APP provides an interface to the sensor node, establishes the communication with the testbed manager and implements the necessary mechanisms for a number of features like timestamping events and coordinating participants.

It uses also the Android's permission dialog to get the participant's permission for his participation to the testbed. In the following subsections we discuss in detail each software module, its functionality, the interconnectivity between them as well as the difficulties and the trade offs we encountered.

5.1.1 Serial Module

The first part that our interface with the sensor node needs is an object that can establish communication, using the smartphone's USB port, with the microprocessor of the connected sensor node. The Serial module is responsible for creating and maintaining this communication. It is the low-level handler of the serial communication between the smartphone and the sensor node. In G-Node sensor node, a FTDI USB controller is used to translate the USB connection of the sensor node to a UART connection, which the MSP430 is equipped with. As a result, the USB controller allows the smartphone to establish a UART communication with the MSP430. The *TxD0* and *RxD0* pins of G-Node's FTDI USB controller (see Figure 5.2) are directly connected to the MSP430 using its UART interface.

The Android Host API allows Android applications to communicate with the connected USB devices. It provides also some additional functions that applications could use in order to execute commands in the USB controller such as setting one of its GPIO pins. However, each USB controller has a different set of commands and command IDs. The application developer should know the set of commands and IDs in order to use the Android Host API. A solution to this problem is to use the API provided by the device drivers of the USB controller. The device drivers use the Android Host API and because they know the available commands in the USB controller are able to provide the necessary functionality to the developer.

In our case, by using the API, which is offered by the drivers of the FTDI USB controller, the Serial module is able to perform the following number of actions:

- open and close the UART connection with the MSP430.
- check if a sensor device is connected.
- write data to, and read data from, the sensor node. The SMoT APP uses a separate thread to read the data from the USB port and a ring buffer data structure with a fixed-size to store them. Using a ring buffer helps us to easily buffer and handle the data.
- configure the UART settings (see Figure 5.1) including the baudrate, the number of data and stop bits and the parity. Configuring properly the settings of the UART connection is an important aspect because both the smartphone and the sensor node should have the same settings in order to communicate.
- set and clear the GPIO pins of the USB controller, which is important for the module that reprograms and resets the sensor node.

Note that, the Serial module is only dependant on the type of the USB controller that the sensor nodes has. Therefore, different USB controllers

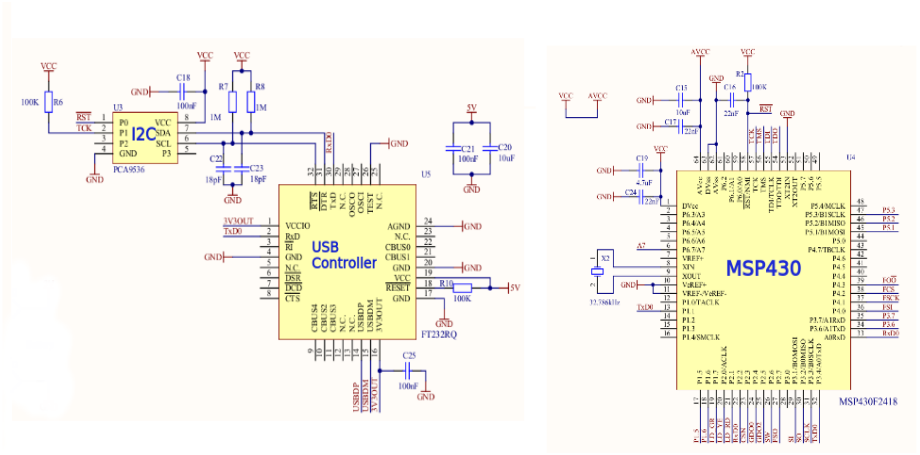


Figure 5.2: GNode hardware organization of USB Controller, I^2C and MSP430.

need different device drives to be used by the SMoT APP. However, the majority of the sensor nodes are equipped with an USB controller from the FTDI family.

5.1.2 BSL Module

Programming the microprocessor of a device requires the flashing of its memory with the desired firmware, which can be done either by triggering its JTAG pins (when available) or by starting its bootloader (if it exists). In our case, the MSP430F2418 microprocessor of the G-Node is equipped with a bootloader, which can be started by triggering the *RST/NMI* and *TCK* pins of the microprocessor as the MSP430 BSL User Guide [8] states. However, depending on how the hardware components of a sensor node are connected there are different ways/paths to access and modify the pins that start the execution of the bootloader. As a result, any time that we would like to add a new sensor node in the testbed, we have to modify the BSL module appropriately in order to have the proper logic to access the microprocessor's pins. As we can see in Figure 5.2, the *RST/NMI* and *TCK* pins of the G-Node's microprocessor are connected to an I^2C circuit and subsequently to the FTDI USB controller. Therefore, the SMoT APP needs to communicate with the I^2C circuit, through the FTDI USB controller, in order to access the microprocessor's bootloader.

Before implementing the BSL module we tried to access the bootloader and program the sensor nodes using an existing script in Python that we executed on the Android smartphone. However, we failed to access the FTDI USB controller because the script needed the appropriate FTDI USB drivers that are not loaded in the default kernel of an Android system. A

solution to this problem is to build a custom Android kernel from scratch including the FTDI USB drivers. However, it will reduce the usability and the ease of deployment of our testbed since we have to install the new custom Android system to every smartphone participating in the testbed. Therefore, we implemented the BSL module, which allows us to interface with the microprocessor of the sensor node and access the bootloader with in an easy and flexible way.

The BSL module is currently, more than 1.5K lines of code and the most complex module of the SMoT APP. It provides the necessary logic and functionality to communicate with the bootloader of the sensor node. However, before the BSL module is able to send commands to the bootloader it needs to invoke the bootloader by applying the bootloader entry sequence, which can be found in [8], to the *RST/NMI* and *TCK* pins of the MSP430F2418 microprocessor. In particular, the BSL module applies the entry sequence to the *P0* and *P1* pins of the I^2C circuit, which are accessible only by using the *RTS* and *DTR* pins of the FTDI USB controller. The BSL module is able to trigger the *P0* and *P1* pins of the I^2C circuit by implementing the I^2C protocol and using the Serial module to access them.

Moreover, the bootloader of the microprocessor has a number of commands that it can execute using a specific data frame protocol, called serial standard protocol (SSP). A 16-bit checksum is calculated over all the transmitted or received bytes in the data frame and some commands are password protected in order to prevent direct data access. In addition, a synchronization character should be sent before sending any commands and an acknowledgement is sent back by the bootloader when a commands is received. The BSL module in the current implementation can send a subset of these commands, which are presented below:

- **Mass erase**: Performs a total erase of the microprocessor's flash memory.
- **SYNC**: Before sending any command, a synchronization character must be sent to maintain internal parameters relevant to the UART and memory timings.
- **Transmit Password**: Some of the bootloader's commands need to receive the bootloader's password before the actual command. The password prevents from actions aiming to harm the sensor node like overwriting the node's firmware.
- **Reset**: Applies the standard reset sequence and the sensor node starts executing the firmware loaded in its flash memory. (Password protected)
- **Bootloader Info**: Gets info for chip's family type and microprocessor's bootloader. (Password protected)

Operation	Baudrate	Stop bits	Data bits	Parity bits
Apply bootloader entry sequence	9600	1	8	Even
Firmware transmission	38400	1	8	Even
Data exchange	115200	1	8	Even

Table 5.1: UART settings for each operation.

- **Baudrate Change**: Sets the baudrate of the communication with the sensor's node bootloader. (Password protected)
- **Transmit Data Block**: This command is important because it allows the BSL module to write a block of data to the flash memory of the microprocessor. The maximum size of data that it can be written with one execution of the command is 250 bytes. (Password protected)

For each command, the BSL module creates the data frame of the command with the checksum and sends first the synchronization character and then the command to the bootloader. Then it waits, without blocking the SMoT APP, for the response from the bootloader and presents it to the participant (if the SMoT APP is running in the background). The response can be three types. First, the command was received and executed successfully, second the command was not valid or was not executed successfully and third a data frame contains the response of the request. An example of the latter is the Bootloader Info command where the BSL module will receive the respective information about the version of the bootloader, the ID of the chip etc.

The sending and receiving of the command data frames is performed using the UART connection, which is handled by the Serial module. Note that the settings of the UART connection are different when we apply the bootloader sequence, transmit the firmware and exchange data (see Table 5.1). Although the parameters for the first two operations are fixed, for the data exchange they can be modified depending on the UART settings that the firmware has set for the sensor node. In addition, we would like to make clear here the main difference of the BSL with the Serial module. The Serial module is a low-level component as it writes and reads whatever information is given to it. The BSL module implements the logic and the protocols of the hardware modules. It knows what and when to send data to the sensor node and to the Serial module, and is able to interpret the received data.

The BSL module also reads and parses the firmware file in order to create its memory image. The firmware has been generated from the researcher by cross-compiling the source file of an experiment. Creating the memory image is not a trivial procedure because the firmware is an object file in ELF format. An ELF file has a specific format and is needed to be read and

timestamp (ms)	sensor output/smartphone event annotations

Table 5.2: Log format.

parsed properly to get the information needed to build the memory image. In order to create the memory image of the firmware, the BSL module creates a representation of the memory contexts by finding the sections, segments and the corresponding memory address of the microprocessors flash memory where the firmware's image will be placed. After creating the proper memory image, the BSL module can finally flash the memory of the microprocessor with the firmware of an experiment. Programming the sensor node is a representative example of the functionality that BSL offers by combining its features. Figure 5.3 presents the complete sequence of the steps that the BSL module performs when it needs to program the sensor node.

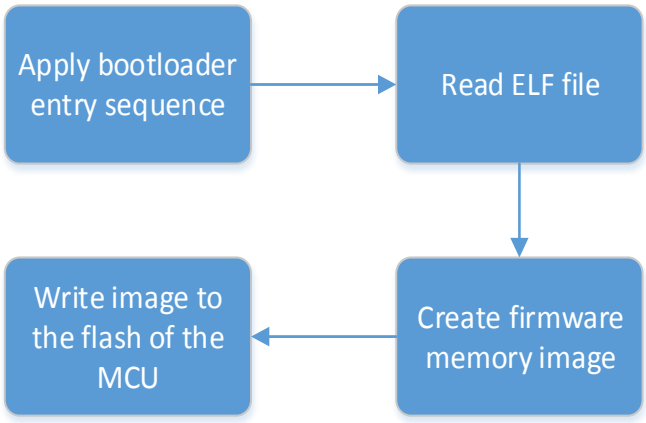


Figure 5.3: BSL module steps for programming a sensor node.

5.1.3 Logger Module

In *SMoT*, both the sensor node and the smartphone can produce data. The sensor node produces the results of an experiment and the smartphone the events generated by the Event manager. These events include notifications with the instructions of a guided experiment, the feedback from the participants and the start and termination of an experiment. The Logger module logs every output of the sensor node and every event from the smartphone in a file stored in the internal flash memory of the smartphone. Every log entry has a specific form specified in Table 5.2.

SMoT utilizes the Network Time Protocol (NTP) [9] of Android to create

the timestamps of a log entry and to provide a global time system for the testbed nodes. Android uses NTP to synchronize the clock of the system with the UTC time with an accuracy that ranges from 1 to 10 milliseconds. However, it needs periodically an internet connection in order to get synchronized with the NTP server. Alternatives exist like the SNTP protocol, which is a simplified version of the NTP protocol, but it can provide an accuracy of 100 milliseconds and GPS, which might not always be available as in the case where the participant is inside a building, but provides a better precision.

5.1.4 Event Manager

The Event manager is the module of the SMoT APP that keeps track of every testbed related event that takes place in the smartphone. For instance, the pressing of a button in the app's interface or the start and termination of an experiment. In addition, it handles any time related event by using timers, which perform an action when they expire. A representative example is the simultaneous start of the testbed nodes. The researcher sets the desirable time and the Event manager of each testbed node, after the reprogramming of the sensor node by the BSL module, starts a timer. When the timer expires, the module gives again the control to the BSL module in order to perform a reset and start the sensor node.

Moreover, the Event manager has the ability to generate events, which can be useful for example to provide a synchronized guidance to the participants of our testbed. In a guided experiment, the Event manager generates and presents the notifications to the participants based on the instructions provided by the researcher. The current version of SMoT APP uses the Android's *Notifications* and *Toast Messages* widgets in order to notify the participants. There are also other ways to notify the participants like using the speaker or the vibration of the phone.

5.1.5 Communication Module

It is essential for the testbed nodes to have an active communication with the testbed manager in order to receive the experiments and send back the results. The Communication module establishes and maintains the connection with the SMoT Server and it is divided in two parts, the File Manager and the Command Manager.

The Command manager implements the client for the Google Cloud Message (GCM) [3] service, which is a cloud messaging service that *SMoT* uses to establish the bidirectional communication between the researcher and the testbed node. We discuss it in detail in Section 5.2. The Command manager runs in the background and listens for commands that are sent by the SMoT Server. It parses the received commands and gives control to the ap-

propriate software module of the SMoT APP, which executes it. After the receiving and completion of any command, the Command manager sends an acknowledgement to the SMoT Server to ensure the correct reception and execution of the command. The current version of the Command manager supports the following sets of commands, which the researcher can issue to control the testbed nodes:

- request the status of a testbed node (*online* or *offline*).
- command a smartphone to program the sensor node and start an experiment with the selected parameters, which we have presented in Table 4.1.
- download a specific file (e.g. the firmware of the sensor node or the file with the instruction of the researcher).
- upload results.
- terminate an experiment. The termination of an experiment implies the termination of the serial communication with the sensor node, which is handled by the Serial module.
- register a new testbed node. The Command manager is responsible for registering a new testbed node to the SMoT Server, but only the first time. Note that this command can be issued only from the SMoT APP and not from the researchers.

The File Manager implements instead the functionality needed to download and store the necessary files of an experiment as well as upload the results to the researcher. *SMoT* uses Dropbox as a cloud file hosting service to store the files of an experiment. Drobox was a suitable choice for two reasons. First, its API is well-integrated on Android and second, in contrast with Google Drive, by using Dropbox the SMoT APP is able to download files from the cloud that have been uploaded by another application or user. However, a Dropbox account is needed in order to have access to the files of an experiment. An FTP server could also be used for downloading and uploading the associated files of an experiment.

5.2 SMoT Server

The SMoT Server is the software part of the testbed that runs on a PC. The current version is more than 1K lines of Java code. It is the part of the testbed that the researchers use in order to access the testbed, send commands and set up their experiments. The software modules of the SMoT Server are three (see Figure 5.4), each providing its own unique functionality.

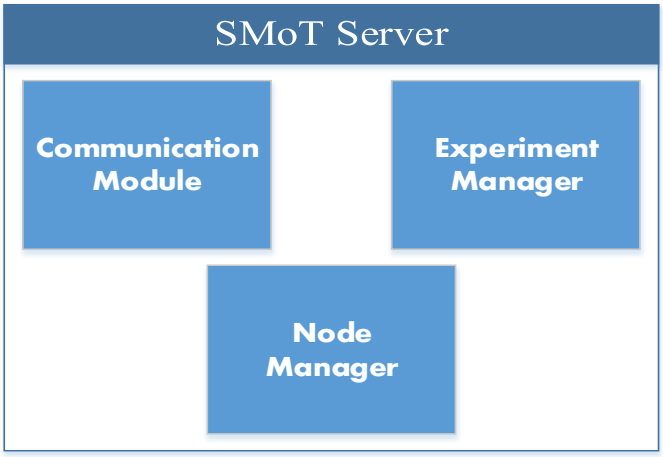


Figure 5.4: SMoT Server software architecture.

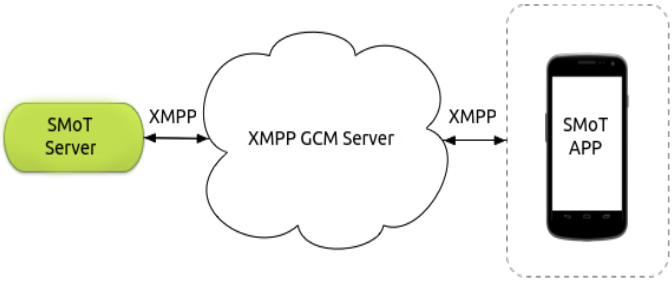


Figure 5.5: GCM implementation in *SMoT*.

5.2.1 Communication Module

The Communication module handles the connection between the SMoT Server and the SMoT APP. *SMoT* uses a cloud messaging service by Google [3] in order allow the SMoT Server to send commands to the testbed nodes. It is a free service that enables the SMoT Server to establish a persistent, asynchronous and bidirectional communication to the SMoT APP without the need to set up our own server like *Pogo* [16]. The Google connection server receives every message from both parties, handles all aspects of queuing of messages and delivers them to the target. It can transfer messages in JSON [6] format with up to 4KB payload, which is adequate for our command messages. An overview of the GCM implementation in *SMoT* can be seen in Figure 5.5.

The GCM service offers two types of connection server, an HTTP and an XMPP server. *SMoT* uses the latter server that relies on the XMPP [13]

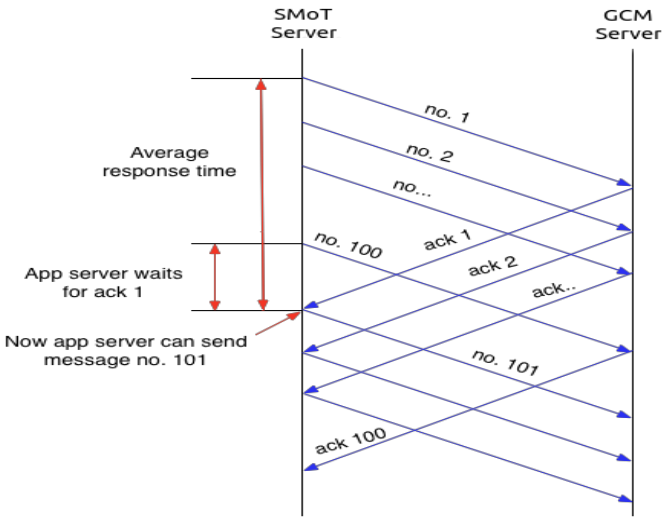


Figure 5.6: Flow of messages from the SMoT Server to the GCM server.

protocol for the communication between the SMoT Server and SMoT APP. XMPP is an instant messaging protocol, which makes it a suitable choice for our testbed because it gives the ability to the SMoT Server to send commands with a delay of a few seconds. In contrast with the HTTP GCM server, the XMPP GCM server gives the ability to the SMoT APP to send messages to the SMoT Server, which is the case of the registration command that is presented in Sub-section 5.1.5. The XMPP GCM server acknowledges every messages it receives and if a problem occurs like a message loss, it will resend it. If a device is offline, the Google server can store the messages for this device for a certain amount of time specified by the developer and delivers the messages as soon as the smartphone is online. A summary of how the acknowledgement mechanism of XMPP GCM server works is presented in Figure 5.6.

If a message is not delivered, it is considered a pending message. The XMPP GCM server can store up to 100 pending messages before dropping every message it receives. Therefore, the SMoT Server implements its own sending and acknowledgement mechanisms. If the number of pending messages exceeds 100, the SMoT Server waits until the XMPP GCM server can accept again new messages and resend it. Moreover, the SMoT Server discards any message that has been acknowledged by the XMPP GCM server, but cannot be delivered. In order to provide a more robust communication with the SMoT APP, the SMoT Server considers a message delivered only if it has received the respective acknowledgement from the SMoT APP and not only from the XMPP GCM server.

5.2.2 Node Manager

The Node Manager is responsible for the management of the registered testbed nodes. It uses an SQLite database to store the registered devices. Each entry of the database stores the unique ID of a testbed node, the manufacturer, the model and the IMEI of the smartphone connected to the testbed node. Android provides full support for SQLite databases, thus the SMoT Server can easily run on Android smartphones instead of PCs. Only minor changes are required and the proper libraries that the application uses. Using the Node manager, the researchers can get information about the details and status of each testbed node and select all or a subset of them in order to run their experiments.

5.2.3 Experiment Manager

The Experiment Manager handles the specification and configuration of an experiment. It prompts the researcher with an interface to specify the name of the firmware and the guidance file, the duration of an experiment and start time as well as the option for the logging of the serial output. It creates the JSON messages with these parameters and use the Communication module to send them to the testbed nodes. Note that in our implementation it is the researcher's responsibility to upload the necessary files of an experiment to Dropbox. The SMoT Server does not send the firmware file of an experiment, but only commands the smartphones to download it from Dropbox. Similar, the researcher takes his results from Dropbox and not from the SMoT Server.

Chapter 6

Evaluation

In this chapter we evaluate *SMoT* on three different aspects: usability, participant coordination and lifetime. In Section 6.1 we quantify the mechanism for the synchronous start of the testbed nodes, which improves the usability by reducing the delays introduced by the communication between the SMoT Server and SMoT APP. In Section 6.2, we present the delays introduced by the participants and show that the *SMoT* mechanisms are able to reduce them and provide an amount of control during an experiment with a sufficient level of realism. Finally, we present the bounds of our testbed in the duration of an experiment in Section 6.3.

6.1 Usability

In *SMoT* we use smartphones, which act as a means to provide the main functionality of a testbed node. However, the usage of smartphones can introduce delays at the execution time of an experiment. The delays are caused by the time a smartphone needs to program a sensor node, which depends on the software and hardware of the smartphone. As a result, the variance at the execution time of an experiment increases between different testbed nodes. To reduce the delays, *SMoT* has a synchronization mechanism that allows the researchers to set the desirable execution time of an experiment and start the testbed nodes simultaneously. We quantified how simultaneous different testbed nodes can execute an experiment by measuring their starting times

The amount of time needed to program and start a sensor node is divided in four events. The first event is the time interval for receiving the command from the SMoT Server. Once the smartphone receives the command it starts programming the sensor node, which is the second event. This event is the most time consuming and we are not able to control it since it depends on the hardware capabilities of the smartphone and on the tasks running at the moment. The third event is the reset of the sensor node, which signals

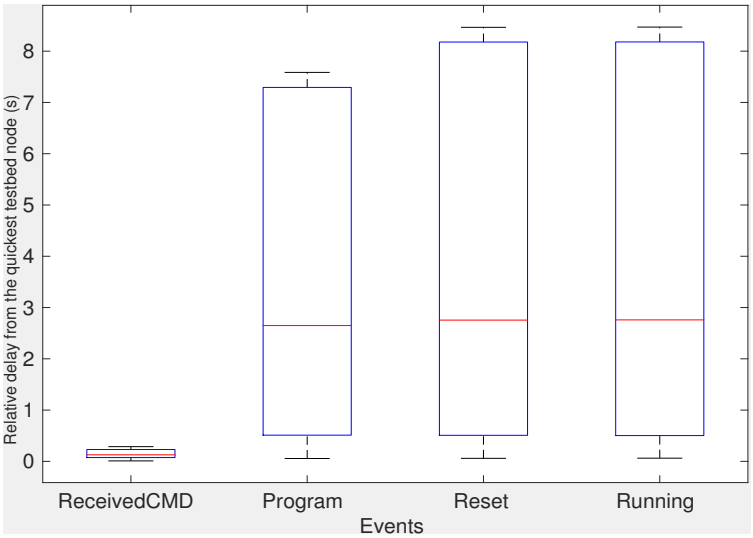


Figure 6.1: Relative delay distributions of the three phones from the quickest phone per event in ten repetitions. Because we do not use the synchronization mechanism of *SMoT* the Running event includes the delays of the ReceivedCMD, Program and Reset events.

the execution of its firmware and the last event is the time the smartphone needs to receive from the sensor node the first serial output of an experiment. We used four testbed nodes with four smartphones, two Samsung Galaxy Nexus running Android 4.4.4 and 4.4.2 respectively, one Samsung GT-I9100 running Android 4.4.4 and a Samsung GT-I9300 running Android 4.3.1 and we measured the time when each event takes place in ten repetitions. At each repetition and event, we computed the relative delays of the three smartphones from the smartphone that performed each event first.

Figure 6.1 shows the distribution of the relative delays of the three smartphones (we exclude the quickest smartphone, which relative delay is always 0) in 10 repetitions when the synchronization mechanism of *SMoT* is not used. We can see that the four phones receive the command from the SMoT Server with a difference of approximately half a second. The use of the XMPP protocol contributes to achieve this bound. However, the time the four smatphones need to program their sensor nodes vary a lot and this variance affects the reset of the sensor nodes and the start of their firmwares since it is accumulated. As a result, the four testbed nodes start the execution of an experiment at different times. In addition, we can observe that the reset operation, which happens after the programming of the sensor node, and the time needed for the sensor to start the firmware are not time consuming operations.

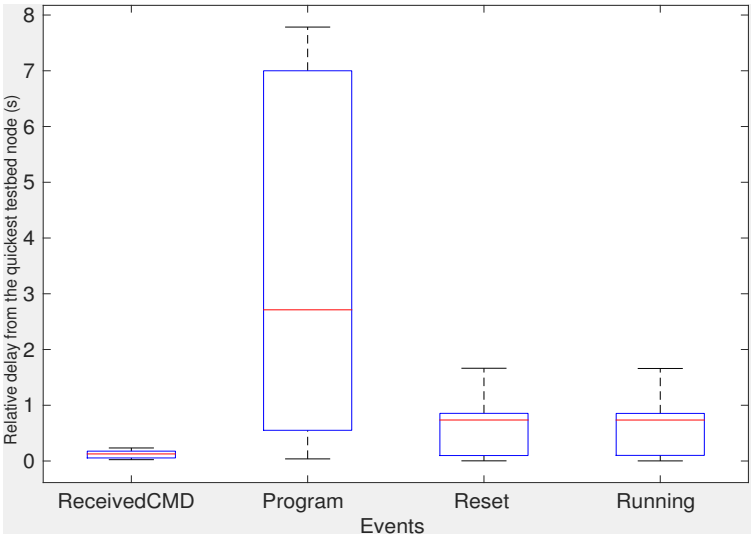


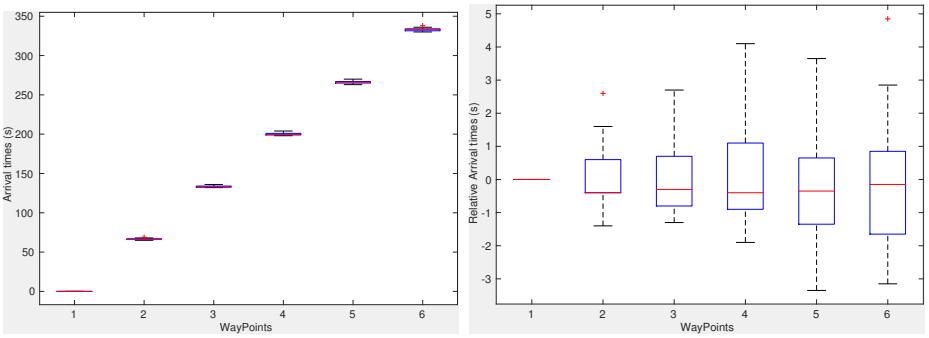
Figure 6.2: Relative delay distribution of the three phones from the quickest phone per event in ten repetitions using the synchronization mechanism of *SMoT*.

As Table 6.1 shows, the reset operation lasts on average roughly one second and the smartphones receive the first output from the sensor after a few milliseconds.

Event	Average duration (sec)
ReceivedCMD	min=0.056, avg=0.199, max=0.368
Program	min=10.639, avg=13.250, max=18.081
Reset	min=0.316, avg=0.558, max=1.182
Running	min=0.301, avg=0.305, max=0.308

Table 6.1: Average time needed for the completion of each event in ten repetitions of four testbed nodes.

On the other hand, Figure 6.2 shows the case where the sensor nodes start simultaneously by using the *SMoT* mechanism. Because the smartphones are synchronized, by using a global time system provided by the SMoT APP, researchers are able to delegate tasks to them in the future. Thus, they are able to set the absolute time that a reset command will be executed simultaneous as soon as all the smartphones finish the programming of their sensor node. Therefore, the absolute time must be set to an arbitrary value that it will be at least later than the time that the slowest phone of the testbed programs its sensor node. Comparing the two figures, we show that the sensor nodes can start almost synchronously with a difference of hundreds milliseconds. Thus *SMoT's* synchronous start mechanism is able to alleviate the uncontrolled delay introduced by the programming of the



(a) Absolute arrival times. (b) Relative arrival times.

Figure 6.3: Distribution of the absolute and relative arrival times at each waypoint of 10 participants in 20 repetitions without using the *SMoT's* guidance mechanism.

sensor nodes.

6.2 Participant Coordination

One of the main goals of *SMoT* is to provide both control over the experiment and realistic conditions. Involving humans makes the control of an experiment difficult since it implies the coordination of the participants of the testbed, which is a challenging task. In addition, human variability introduces delays in the coordination of the participants because each persons walks for example at different speeds. Nevertheless, using *SMoT*, researchers are able to run an experiment in realistic conditions (e.g. human mobility) while having a sufficient amount of control over it.

To evaluate *SMoT's* coordination capabilities, we examined how its guidance, feedback and logging mechanisms affect the execution of an experiment. We run an experiment in a corridor of our building with 10 participants and 20 repetitions (2 repetitions per participant). We set up 6 waypoints, specific spots on the floor, with 4 meters distance between each other where the participants moved and stood for 60 seconds. At each waypoint we marked the arrival time of each participant in each repetition by using a stopwatch.

Figure 6.3(a) shows the distribution of the arrival times at each waypoint. We can see that as the participants were moving along the waypoints the variance in the arrival times was increasing. The reason is the human variability since the participants were moving with different speeds. Some of the participants were arriving at a waypoint earlier or later than others as it can be seen in Figure 6.3(b), which presents the relative arrival times at each waypoint. The relative arrival time is the result of the subtraction

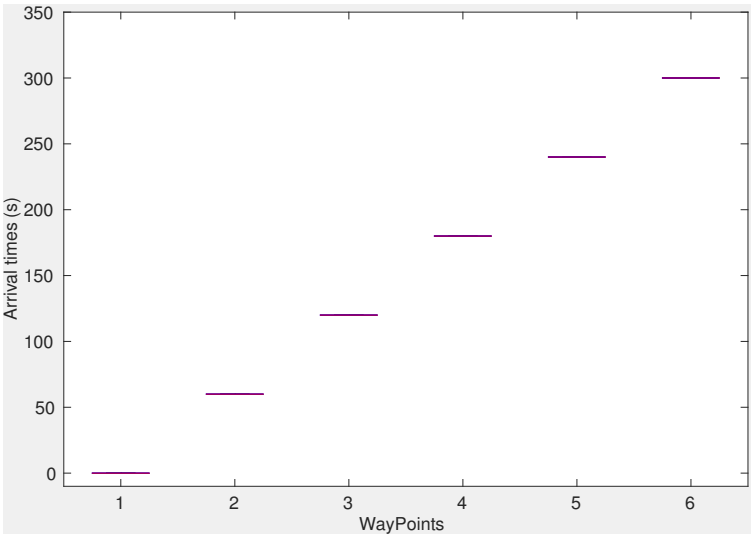


Figure 6.4: Arrival time after SMoT guidance, feedback and logging mechanisms.

of the mean arrival time (of all repetitions) from the absolute arrival time at a waypoint.

Figure 6.4 presents the arrival times at each experiment if we do not take into consideration the movement between the waypoints. By using the *SMoT's* guidance, feedback and logging mechanisms we can align the logging data and the arrival times to those in Figure 6.4. Despite the fact that during the experiment the participants arrived to the waypoints at different times, the testbed mechanisms enable us to know when a participants has left and arrived at a waypoint and as a result to correlate the logging data with the proper waypoint and arrival times.

Until now we showed the delays introduced by the smartphones of the testbed nodes and the participants of the testbed. We analysed and quantified the delays and we presented how the mechanisms of our testbed can reduce them. By reducing the delays of an experiment using *SMoT's* mechanism for coordination, feedback collection and logging we give to the researcher the ability to have more control over his experiments, which in turn improves the consistency of his results.

We evaluated how the control over an experiment can affect the consistency of the results by running three experiments and comparing three different control scenarios. For our experiments we used *The Rack* [34], a static testbed that consists of 100 nodes located in the ceiling of the 9^{th} floor of the EWI faculty building at the Technical University of Delft. During each experiment the participant, who was holding a testbed node, was moving in one direction towards 6 predefined waypoints, stopping at each waypoint for 60 seconds. The participant's testbed node was communicat-

(a) (b)

Figure 6.5: The trolley standing in a waypoint in a fixed position (black tape in the ground) while was carrying in fixed orientation a *SMoT* testbed node.

ing wirelessly with the nodes of *The Rack* and was executing an algorithm that was collecting and writing to the serial the IDs of the neighbors at each waypoint.

For each scenario, we computed the similarity between the neighborhood cardinality at each waypoint over 10 different repetitions. To compare the different repetitions we used the Jaccard Index [5] because of its simplicity and the fact that we had sets, which represent the neighbors at each waypoint. However, it is possible to use another similarity metric.

The first scenario provides a fully controlled but non-realistic environment since we used a trolley (to emulate a robot) instead of a human. The trolley was carrying one testbed node in a fixed orientation (see Figure 6.5). We moved the trolley along the waypoints and placed it in a fixed position (marked by tape on the floor). We used *SMoT's* guidance, feedback and annotation mechanisms in order to collect and correlate the results to the correct waypoints. In Figure 6.6, we present the distribution of the similarities between different repetitions of the experiment at each waypoint. The only factor that affects the experiment here, is the wireless link characteristics that were fluctuating due to changes in the environment and people moving around the offices. Thus, this is the best similarity we can achieve. Note that at waypoint 6 the similarity is constant due to the fact that the half side of the ceiling do not have The Rack's nodes because it is where the heating systems of the floor is located.

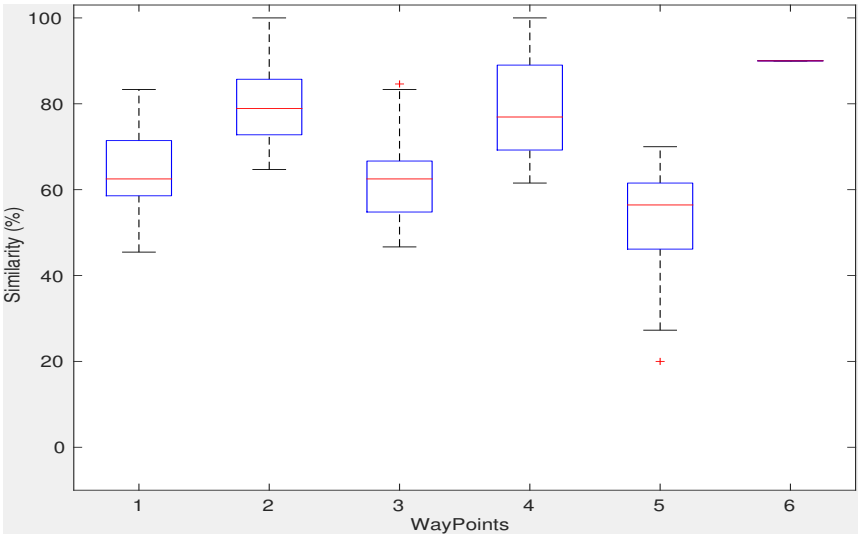


Figure 6.6: Distribution of similarities between 10 repetitions at each waypoint using a trolley.

In the second scenario, we used a person for the experiment but without any guidance from the testbed. This resulted in a realistic, but uncontrolled environment. We gave a testbed node to a participant who was moving towards the waypoints and stood at each of them for 60 seconds. It was the participant's responsibility to keep track of the time and decide when to move to the next waypoint. In contrast to the previous experiment, Figure 6.7 shows that the similarity between the repetitions of the experiment at each waypoint is very low because we did not have control during the experiment. The results of the experiment did not have any annotations about the time that the participant left or arrived at a waypoint, but only the serial output of the sensor node with its timestamp. Thus, we could not correlate the correct logging data with the appropriate waypoint and we split the results in portions of 60 seconds (the time that the participant stood at each waypoint). As a consequence, the logging data that corresponds to each waypoint could include data that belong to the previous waypoint and to the physical space between them.

In the third scenario, the participant was moving using the guidance and the feedback mechanisms of *SMoT*. This way, we were able to achieve realistic conditions, exposing the protocols to human characteristics while providing a good amount of control. The similarity results can be seen in Figure 6.8. The distribution of the similarities between repetitions is smaller than without guidance, with a mean value that is similar but lower than the trolley experiments. By guiding the participant, collecting his feedback and annotating each event in the results we could distinguish which logging

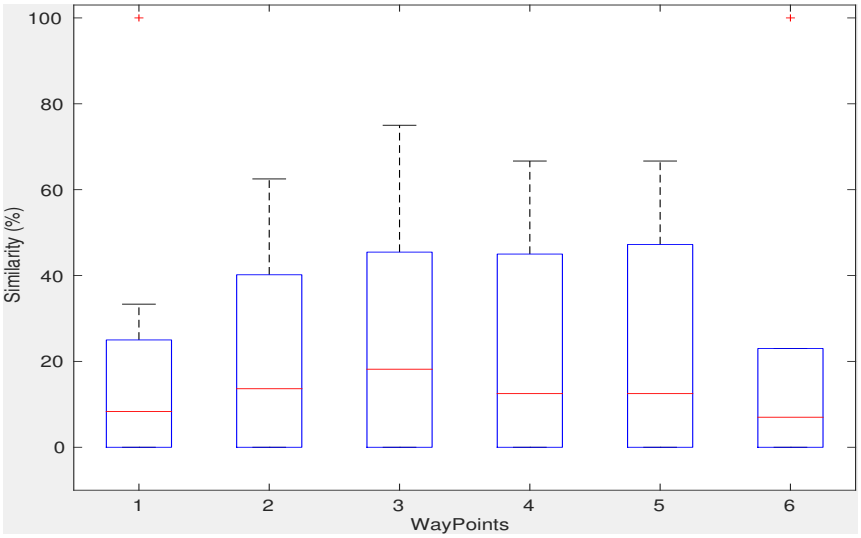


Figure 6.7: Distribution of similarities between 10 repetitions at each waypoint using a participant and *SMoT* without guidance.

data belong to which waypoint and as a result have better control of our experiment. There was no need for extra assistance to keep track of the participant's actions since the smartphone of the testbed node performed that. Comparing Figures 6.7 and 6.8, we can see that our testbed with its mechanisms helps researchers to obtain an amount of control over their experiments, which assists the collection and understanding of their results.

Human Factor

In the last two scenarios, we exposed the application to realistic conditions since the participant was the subject that was carrying the testbed node and was moving every time with different orientation and stand position at each waypoint. The human diversity, like the movement speed or the orientation of the testbed node, was exposed to the application while debugging and collection of the results were feasible with low effort and time.

Comparing Figure 6.6 with Figure 6.8, we show that the human factor and especially the human variability can affect the protocol or the application since the only difference between these two experiments is the use of a person for carrying the testbed node. In addition, even with perfect coordination, we show in Figure 6.8 that the results are not identical but similar. That is, our testbed can capture the consequences of human variability, and is able to expose the application or protocol to human characteristics.

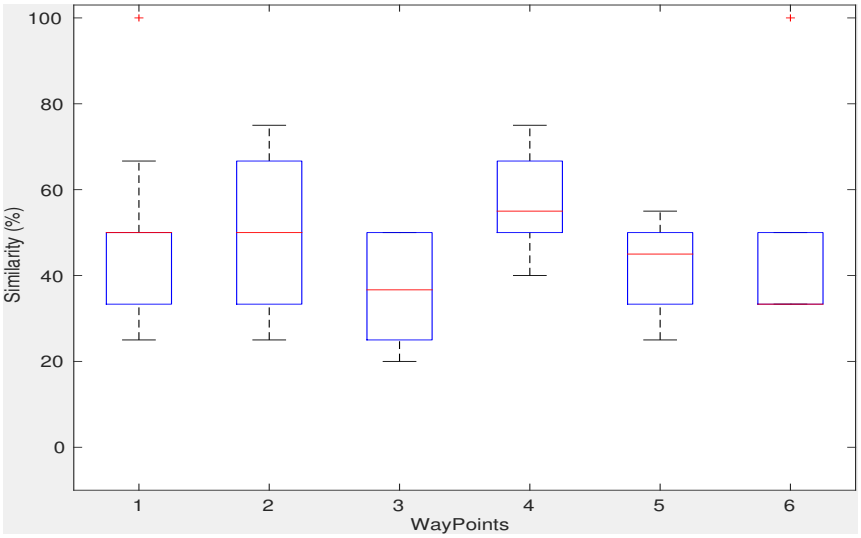


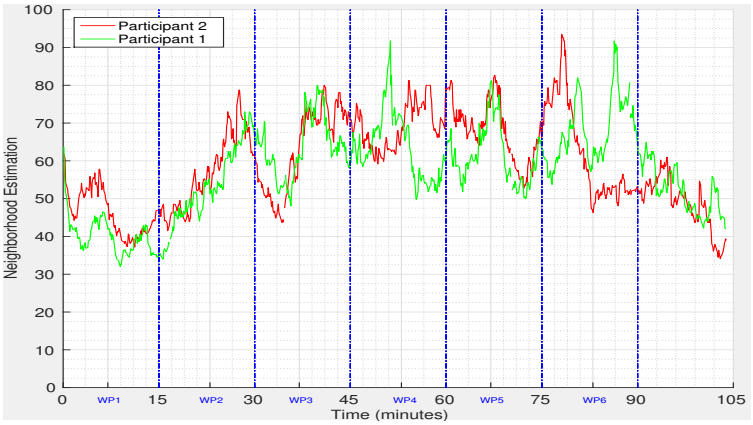
Figure 6.8: Distribution of similarities between 10 repetitions at each waypoint using a participant and *SMoT* with the guidance and feedback mechanisms.

Real Case - Estreme

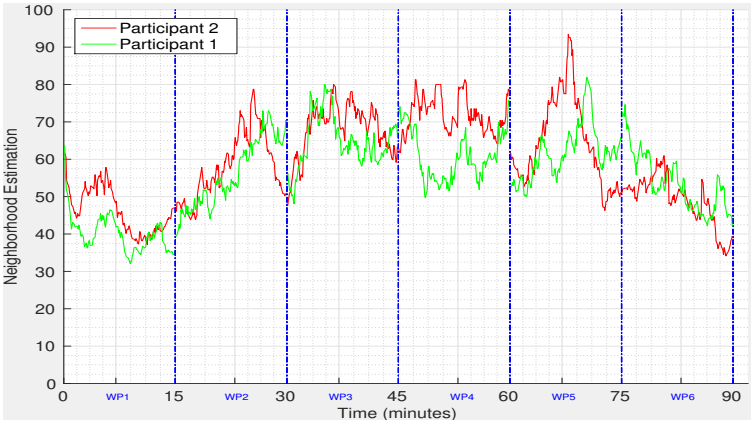
We testbed *SMoT* in a different set up using a real-world protocol for neighborhood cardinality estimation, named *Estreme* [18]. *Estreme* is an estimator for dynamic wireless networks. That is, networks that are dense (up to one hundred of neighbors) and mobile as in the case of crowds. *Estreme* is able to estimate in a concurrent, fast, asynchronous and accurate manner the number of the nodes that a node has in its neighborhood. In addition, the experiment consisted of two participants carrying each a testbed node who were moving towards six predefined waypoints with 8 meters distance between them and stood at each waypoint for 15 minutes. At each waypoint we wanted to collect the estimation of the protocol running on the sensor nodes.

Since the improvements of SMoT over the participant coordination are very similar to the ones in the previous section, we do not present the results of the experiment but we give a brief description.

Each participant arrived to each waypoint at different times with a distribution of the arrival times that is similar to Figure 6.3(a). Moreover, by using *SMoT* to guide the participants and collect their feedback we were able to know when the participants left or arrived to a waypoint, since there were annotations in the logging data added by the SMoT APP. Thus, *SMoT* helped us to get consistent results with the estimations of the protocol from each participants at each waypoint.



(a) Before parsing with SMoT.



(b) After parsing with SMoT.

Figure 6.9: *SMoT's* coordination, feedback collection and logging mechanisms improves the consistency of the results.

In figures 6.9(a) and 6.9(b), we show the estimation of *Estreme* before and after parsing them with *SMoT's* logging annotations and timestamps. We can see that if we do not parse the logging data with *SMoT*, the duration of our data is more than 90 minutes. However, 90 minutes is the correct duration of the experiment since we had 6 waypoints and the participants stood at each of them for 15 minutes. In addition, the data related to each waypoint is not the proper because each waypoint (except the first) has data that belongs to the previous or the distance in between them. On the other hand, using *SMoT's* logging annotations and timestamps we are able to assign the data to the correct waypoints and discard the data between the waypoints.

6.3 Lifetime

By understanding and analysing the lifetime of *SMoT*, we are able to set some boundaries to the maximum duration of an experiment and avoid situations where the participants will leave the experiments because our infrastructure drained the battery of their phones. Moreover, the researchers can organize and set up their experiments better, based on the amount of time available without intermediate unnecessary stops and pauses. The maximum duration of an experiment depends on the available energy of the participant's smartphone battery, which is the main source of power for both the smartphone and the sensor node.

To understand the energy requirements of *SMoT*, we measured the power consumption of a smartphone using the Monsoon Power Monitor [7] tool, which provides a comprehensive and easy to use interface for this purpose. Our experiments were conducted with three different smartphones, a Samsung Galaxy S II, a Samsung Galaxy S III and a Samsung Galaxy Nexus running Android 4.4.4, 4.3.1 and 4.4.4, respectively. We measured the power consumption in five different cases:

- IDLE – The sensor node is not connected to the smartphone and the phone has its screen off and no applications are running.
- USB OTG – Only the USB OTG connector is attached to the smartphone without the sensor node.
- SENSOR – The sensor node is connected to the smartphone using the USB OTG cable.
- EXP. – The sensor node is connected and the SMoT APP is running in the background an experiment without guidance. It is reading the logging data from the serial and writing them on a file in the flash memory of the phone.
- EXP.+GUIDANCE – The SMoT APP is running in the foreground an experiment with guidance. Thus, it performs the same operations as in the EXP. case plus the screen is always on (in the lowest brightness) in order to collect the participant's feedback and guide them with notifications.

Figure 6.10 presents the power consumption of each phone for each of the above cases as well as the estimation of the lifetime of the testbed node, and as a result the estimation of the maximum duration of an experiment. The lifetime is related to the battery capacity of the phone, so it is different for each phone and case, see the legend in Figure 6.10.

A number of important observations can be seen in Figure 6.10. First, we can observe that the power consumption increases drastically when the

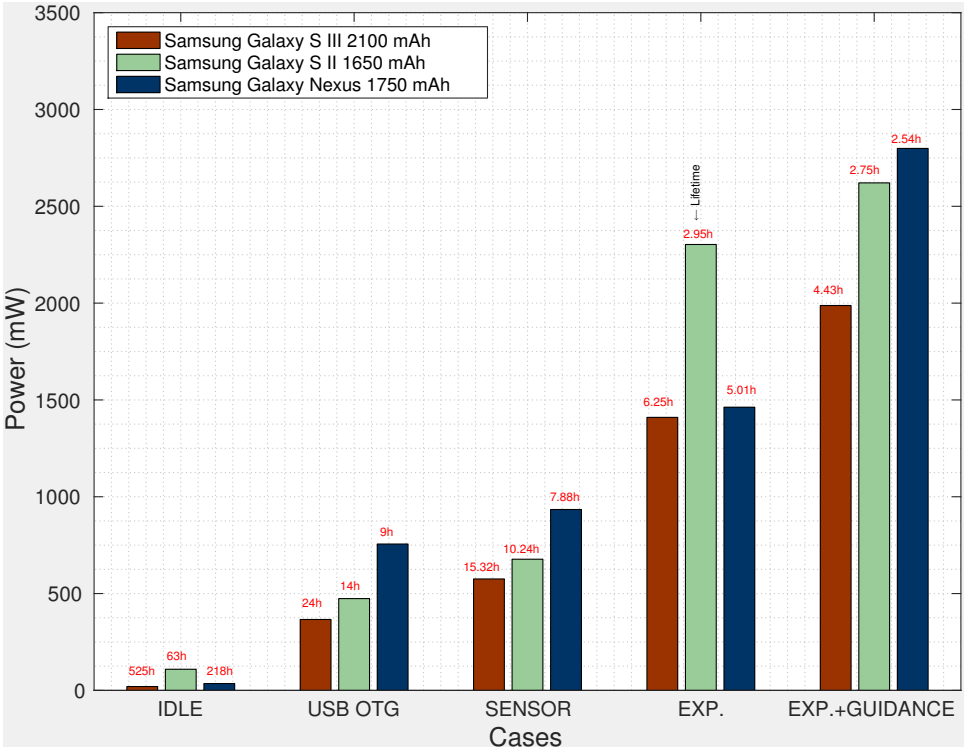


Figure 6.10: Smartphones power consumption.

USB OTG cable is connected to the phone, without the sensor node. This is a limitation of the USB On-The-Go mechanism, which has a special circuit that allows it to operate either in the normal USB mode or in the USB OTG mode. As we already mentioned in Chapter 5 we can alternatively power the sensor node directly from the battery of the phone and use the USB OTG connection only to program the sensor node and log data when it is needed. Powering directly the sensor node will increase the life time of our testbed while the sensor node is continuously active and can run experiments. Furthermore, connecting the sensor node does not increase the power consumption significantly.

Second, we can see that in the EXP. case in which the SMoT APP is running, there is a big increase in the power consumption in all the phones. When the SMoT APP is running an experiment, two operations take place constantly. The serial connection of the smartphone is always active because the sensor node sends its results to the smartphone. As a result, reading from the serial prevents the CPU of the smartphone from going to sleep. The smartphone also writes the logging data, which it receives from the sensor node, to its flash memory.

We measured the power consumption of the SMoT APP when it writes data to the flash memory of the smartphone without reading from the serial,

and the operation costs a few mW at each phone. In addition, we measured the power consumption of the SMoT APP when it reads continuously data from the serial without writing to the flash memory of the smartphone, and we noticed that this operation consumes hundreds of mW. Therefore, it is the main reason for the big increase in the power consumption.

Moreover, depending on the hardware of the smartphone and the Android version, the increase in the power consumption when the SMoT APP is running varies a lot. For instance, in the case of the Samsung Galaxy SII, which is the oldest of the three phones, the increase in the power consumption is higher compared to the other two smartphones. After performing a number of measurements with Monsoon Power Monitor, we could not understand what is the main source of the big increase in the specific smartphone. We assume that the different kernel version from the Samsung Galaxy Nexus, which runs the same version of Android, and the way the hardware manipulates the serial connection are the main causes for this big increase.

Finally, Figure 6.10 shows that running an experiment with guidance can be very expensive in terms of power. The guidance decreases the lifetime of the testbed and shortens the duration of an experiment because the screen, which is one of the most power consuming components of a smartphone, is always on in order to guide the participants and collect their feedback. We can see that for some cases like the Samsung Galaxy Nexus, the duration of an experiment can be reduced by half because its screen is consuming a lot. On the other hand, in the case of the Samsung Galaxy SII the screen does not consume a lot of power due to the fact that it has the smallest screen size of the three phones and the lowest resolution, which is three times smaller than the resolution of the other two phones.

Note that we measured also the power consumption of the phone when the SMoT APP programs a sensor node, downloads the files of an experiment and uploads the results. Nevertheless, we do not present these results here because these operations last for a few seconds and they do not significantly affect the life time of a testbed node.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Wireless sensor networks have recently started to become valuable solutions for a variety of applications that involve humans. Some representative examples are human-activity recognition and crowd monitoring. Researchers design and implement protocols and applications for this kind of networks, taking also into consideration the human factors such as the body orientation and moving speed. Before the final deployment the functionality and correctness should be tested both in simulation and in real-world experiments.

For these reasons, in this thesis, we designed and implemented *SMoT*, a novel testbed facility for human-centric wireless sensor networks. *SMoT* consists of participants who each carry a testbed node, which is a sensor node attached to a smartphone. The sensor node is the component where the experiments run and the smartphone, which runs the SMoT APP, acts as a mean to provide the functionality and the power of the testbed node.

The SMoT APP interfaces with the sensor node, communicates with the researcher, programs the sensor nodes, guides the participants and collects their feedback. The researchers are able to access the testbed, setup and run an experiment by using the SMoT Server, which communicates with the testbed nodes, sends commands, delegates tasks and collects the results. With that way, *SMoT* exposes the WSN applications to the human factors, while giving the researchers a platform where they are able to test and verify the correctness and the behaviour of their designs in a realistic and controlled environment.

Before implementing *SMoT*, we defined the appropriate set of requirements that a testbed for human-centric WSNs should fulfil. We evaluated how *SMoT* fulfils these requirements and examined the efficacy of its mech-

anisms and the overall usability of the testbed. Finally, we showed how the testbed's mechanisms help providing realistic conditions while guaranteeing an amount of control over an experiment. This makes running experiments faster, coordinating the participants simpler and the collected results more understandable and consistent.

7.2 Future Work

SMoT is a fully functional testbed, able to run experiments for human-centric wireless sensor network research. However, there are some limitations and some improvements that can be applied in the future.

- The lifetime of a testbed node is an important aspect for our testbed. The current version of *SMoT* has limitations in the duration of an experiment. The sensor node drains the battery of the smartphone even when it is not running an experiment. As a result, a possible solution is to modify the USB connector in order to be able to switch between the normal USB and USB OTG mode. We already have a PCB with a USB controller and an audio jack but the integration with the SMoT APP is a work in progress.

  Another alternative is to power the sensor node with the smartphone's battery and use the USB connection only to reprogram and log data from the sensor node. In that case the sensor node will be continuously active and we will use the USB connection when needed. This will also extend the lifetime of our testbed and the duration of an experiment. However, we should evaluate both solutions and analyse the power consumption before the final integration with the testbed.

- In order to estimate the lifetime of a testbed node we have to measure the power consumption of its smartphone. In this thesis, we measured the power consumption of three specific phones. If we would like to use a new smartphone as part of a testbed node, we have to measure again its power consumption. To avoid performing measurements for each phone, the development of a model that it will estimate the available lifetime of a testbed node depending on the available energy of the smartphone is necessary.

- In the current implementation of *SMoT*, the testbed nodes support only one sensor node, the G-301 from SOWNet. There is a wide variety of sensor nodes that are used for wireless sensor networks. Therefore, the support of more sensor nodes by the testbed would make the testbed applicable to more protocols and applications, since each of them is designed for a specific sensor node. In addition, it would make the testbed heterogeneous, which is the case for some scenarios.

Bibliography

[1] Apple app store. http://store.apple.com/.

[2] Future technology devices international. http://www.ftdichip.com/.

[3] Google cloud messaging cloud connection server (xmpp). https://developer.android.com/google/gcm/ccs.html.

[4] Google play store. http://play.google.com.

[5] Jaccard index. http://ag.arizona.edu/classes/rnr555/lecnotes/10.html.

[6] Json. http://www.json.org/.

[7] Monsoon power monitor. https://www.msoon.com/LabEquipment/PowerMonitor/.

[8] Msp430 programming via the bootstrap loader(bsl). http://www.ti.com/lit/ug/slau319j/slau319j.pdf.

[9] Network time protocol. http://www.ntp.org/.

[10] Roomba discovery. http://www.irobot.com/.

[11] Sownet g-node g301. https://www.sownet.nl/index.php/products/gnode.

[12] Usb on-the-go. http://www.usb.org/developers/onthego/.

[13] The xmpp standards foundation. http://xmpp.org.

[14] C.A. Boano, M. Zuniga, J. Brown, U. Roedig, C. Keppitiyagama, and K. Romer. Templab: A testbed infrastructure to study the impact of temperature on wireless sensor networks. In *Information Processing in Sensor Networks, IPSN-14 Proceedings of the 13th International Symposium on*, pages 95–106, April 2014.

[15] S. Bromage, C. Engstrom, J. Koshimoto, M. Bromage, S. Dabideen, M. Hu, R. Menchaca-Mendez, D. Nguyen, B. Nunes, V. Petkov, D. Sampath, H. Taylor, M. Veyseh, J. J. Garcia-Luna-Aceves, K. Obraczka, H. Sadjadpour, and B. Smith. Scorpion: A heterogeneous wireless networking testbed. *SIGMOBILE Mob. Comput. Commun. Rev.*, 13(1):65–68, June 2009.

[16] Niels Brouwers and Koen Langendoen. Pogo, a middleware for mobile phone sensing. In *Proceedings of the 13th International Middleware Conference*, Middleware '12, pages 21–40, New York, NY, USA, 2012. Springer-Verlag New York, Inc.

[17] Marco Cattani, Ştefan Gună, and Gian Pietro Picco. Group monitoring in mobile wireless sensor networks. In *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, pages 1–8. IEEE, 2011.

[18] Marco Cattani, Marco Zuniga, Andreas Loukas, and Koen Langendoen. Lightweight neighborhood cardinality estimation in dynamic wireless networks. In *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, IPSN '14, pages 179–189, Piscataway, NJ, USA, 2014. IEEE Press.

[19] Ioannis Chatzigiannakis, Stefan Fischer, Christos Koninis, Georgios Mylonas, and Dennis Pfisterer. Wisebed: an open large-scale wireless sensor network testbed. In *Sensor Applications, Experimentation, and Logistics*, pages 68–87. Springer, 2010.

[20] Cory Cornelius, Apu Kapadia, David Kotz, Dan Peebles, Minho Shin, and Nikos Triandopoulos. Anonysense: Privacy-aware people-centric sensing. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, pages 211–224, New York, NY, USA, 2008. ACM.

[21] Tathagata Das, Prashanth Mohan, Venkata N. Padmanabhan, Ramachandran Ramjee, and Asankhaya Sharma. Prism: Platform for remote sensing using smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 63–76, New York, NY, USA, 2010. ACM.

[22] Pradipta De, Ashish Raniwala, Rupa Krishnan, Krishna Tatavarthi, Jatan Modi, Nadeem Ahmed Syed, Srikant Sharma, and Tzi cker Chiueh. Mint-m: An autonomous mobile wireless experimentation platform. In *Proc. of Mobisys 2006*, pages 124–137, 2006.

[23] Manjunath Doddavenkatappa, MunChoon Chan, and A.L. Ananda. Indriya: A low-cost, 3d wireless sensor network testbed. In Thanasis Korakis, Hongbin Li, Phuoc Tran-Gia, and Hong-Shik Park, editors, *Testbeds and Research Infrastructure. Development of Networks and Communities*, volume 90 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 302–316. Springer Berlin Heidelberg, 2012.

[24] E. Egea-Lopez. Simulation tools for wireless sensor networks. In *SPECTS*, 2005.

[25] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo. A survey on facilities for experimental internet of things research. *Communications Magazine, IEEE*, 49(11):58–67, November 2011.

[26] Osman Khalid and Muhammad Sualeh. Comparative study on mobile wireless sensor network testbeds. *International Journal of Computer Theory & Engineering*, 5(2), 2013.

[27] Roman Lim, Federico Ferrari, Marco Zimmerling, Christoph Walser, Philipp Sommer, and Jan Beutel. Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*, IPSN '13, pages 153–166, New York, NY, USA, 2013. ACM.

[28] Claudio Martella, Maarten van Steen, Aart Halteren, Claudine Conrado, and Jie Li. Crowd textures as proximity graphs. *Communications Magazine, IEEE*, 52(1):114–121, 2014.

[29] Anandatirtha Nandugudi, Anudipa Maiti, Taeyeon Ki, Fatih Bulut, Murat Demirbas, Tevfik Kosar, Chunming Qiao, Steven Y. Ko, and Geoffrey Challen. Phonelab: A large programmable smartphone testbed. In *Proceedings of First International Workshop on Sensing and Big Data Mining*, SENSEMINE'13, pages 4:1–4:6, New York, NY, USA, 2013. ACM.

[30] Olof Rensfelt, Frederik Hermans, Per Gunningberg, Lars-Ake Larzon, and Erik Bjornemo. Repeatable experiments with mobile nodes in a relocatable wsn testbed. *The Computer Journal*, page 14, 2011.

[31] Hugues Smeets, Chia-Yen Shih, Marco Zuniga, Tobias Hagemeier, and Pedro Jos Marrn. Trainsense: A novel infrastructure to support mobility

in wireless sensor networks. In Piet Demeester, Ingrid Moerman, and Andreas Terzis, editors, *EWSN*, volume 7772 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 2013.

[32] L.P. Steyn and G.P. Hancke. A survey of wireless sensor network testbeds. In *AFRICON, 2011*, pages 1–6, Sept 2011.

[33] A.-S. Tonneau, N. Mitton, and J. Vandaele. A survey on (mobile) wireless sensor network experimentation testbeds. In *Distributed Computing in Sensor Systems (DCOSS), 2014 IEEE International Conference on*, pages 263–268, May 2014.

[34] M. Woehrle, M. Bor, and K. Langendoen. 868 MHz: A noiseless environment, but no free lunch for protocol design. In *Networked Sensing Systems (INSS), 2012 Ninth International Conference on*, pages 1–8, June 2012.

[35] Allen Y. Yang, Roozbeh Jafari, S. Shankar Sastry, and Ruzena Bajcsy. Distributed recognition of human actions using wearable motion sensor networks, 2009.