Dynamic Deployment of Sensing Experiments in the Wild Using Smartphones

Nicolas Haderer, Romain Rouvoy, and Lionel Seinturier

Inria Lille – Nord Europe,
LIFL - CNRS UMR 8022,
University Lille 1, France
firstname.lastname@inria.fr

Abstract. While scientific communities extensively exploit simulations to validate their theories, the relevance of their results strongly depends on the realism of the dataset they use as an input. This statement is particularly true when considering human activity traces, which tend to be highly unpredictable. In this paper, we therefore introduce *APISENSE*, a distributed crowdsensing platform for collecting realistic activity traces. In particular, *APISENSE* provides to scientists a participative platform to help them to easily deploy their sensing experiments in the wild. Beyond the scientific contributions of this platform, the technical originality of *APISENSE* lies in its Cloud orientation and the dynamic deployment of scripts within the mobile devices of the participants.We validate this platform by reporting on various crowdsensing experiments we deployed using Android smartphones and comparing our solution to existing crowdsensing platforms.

1 Introduction

For years, the analysis of activity traces has contributed to better understand crowd behaviors and habits [13]. For example, the *Urban Mobs* initiative[1] visualizes SMS or call activities in a city upon the occurrence of major public events. These activity traces are typically generated from GSM traces collected by the cellphone providers [21]. However, access to these GSM traces is often subject to constraining agreements with the mobile network operators, which restrict their publication, and have a scope limited to telecom data.

In addition to that, activity traces are also used as a critical input to assess the quality of scientific models and algorithms. As an example, the *Reality Mining* activity traces[2] collected by the MIT Media Lab or the *Stanford University Mobile Activity TRAces* (SUMATRA)[3] have become a reference testbed to validate mobile algorithms in ad hoc settings [18]. The *Community Resource for Archiving Wireless Data At Dartmouth* (CRAWDAD)[4] is another initiative from the Dartmouth College aiming at building a repository of wireless network traces. Nonetheless, the diversity of the activity traces

[1] http://www.urbanmobs.fr
[2] http://reality.media.mit.edu
[3] http://infolab.stanford.edu/pleiades/SUMATRA.html
[4] http://crawdad.cs.dartmouth.edu

J. Dowling and F. Taïani (Eds.): DAIS 2013, LNCS 7891, pp. 43–56, 2013.

available in these repositories remains limited and thus often constrains scientists to tune inadequate traces by mapping some of the parameters to their requirements. More recently, some approaches have mined the data exposed by location-based social network like Gowalla or Foursquare, but the content of these activity traces remains limited to coarse-grained locations collected from users check-ins.

In this context, we believe that cellphones represent a great opportunity to collect a wide range of crowd activity traces. Largely adopted by populations, with more than 472 millions sold in 2011 (against 297 millions in 2010) according to Gartner institute[5], smartphones have become a key companion in people's dailylife. Not only focusing on computing or communication capabilities, modern mobile devices are now equipped of a wide range of sensors enabling scientist to build a new class of datasets. Furthermore, the generalization of *app stores* or *markets* on many mobile phone platforms leverages the enrollment of participants to a larger scale than it was possible previously.

Using cellphones to collect user activity traces is reported in the literature either as *participatory sensing* [4], which requires explicit user actions to share sensors data, or as *opportunistic sensing* where the mobile sensing application collect and share data without user involvement. These approaches have been largely used in the multiples research studies including traffic and road monitoring [2], social networking [15] or environmental monitoring [16]. However, developing a sensing application to collect a specific dataset over a given population in not trivial. Indeed, a participatory and opportunistic sensing application needs to cope with a set of key challenges [6,12], including energy limitation, privacy concern and needs to provide incentive mechanisms in order to attract participants.

These constraints are making difficult, for scientists non expert in this field, to easily collect realistic datasets for their studies. But more importantly, the developed ad hoc applications may neglect privacy and security concerns, resulting in the disclosure of sensible user information. With regards to the state-of-the-art in this field, we therefore observe that current solutions lack of reusable approaches for collecting and exploiting crowd activity traces, which are usually difficult to setup and tied to specific data representations and device configurations. We therefore believe that crowdsensing platforms require to evolve in order to become more open and widely accessible to scientific communities. In this context, we introduce *APISENSE*, an open platform targeting multiple research communities, and providing a lightweight way to build and deploy opportunistic sensing applications in order to collect dedicated datasets.

This paper does not focus on user incentive challenges, which we already addressed in [10] to provide an overview of appropriate levers to encourage scientists and participants to contribute to such sensing experiments.

The remainder of this paper is organized as follows. We provide an overview of the *APISENSE* platform by detailling the server-side infrastructure as well as the client-side application (cf. Section 2). Then, we report on the case studies we deployed in the wild using *APISENSE* (cf. Section 3) before comparing our solution to the state-of-the-art approaches (cf. Section 4). Finally, we discuss the related work in this domain (cf. Section 5) before concluding (cf. Section 6).

[5] `http://www.gartner.com/it/page.jsp?id=1924314`

2 Distributed Crowdsensing Platform

The *APISENSE* platform distinguishes between two roles. The former, called *scientist*, can be a researcher who wants to define and deploy an experiment over a large population of mobile users. The platform therefore provides a set of services allowing her to describe experimental requirements in a scripting language, deploying experiment scripts over a subset of participants and connect other services to the platform in order to extract and reuse dataset collected in other contexts (*e.g.*, *visualization*, *analysis*, *replay*). Technically, the server-side infrastructure of *APISENSE* is built on the principles of Cloud computing in order to offer a modular service-oriented architecture, which can be customized upon scientist requirements. The latter is the mobile phone user, identified as a *participant*. The *APISENSE* platform provides a mobile application allowing to download experiments, execute them in a dedicated sandbox and automatically upload the collected datasets on the *APISENSE* server.

2.1 Server-Side Infrastructure

The main objective of *APISENSE* is to provide to scientist a platform, which is open, easily extensible and configurable in order to be reused in various contexts. To achieve this goal, we designed the server-side infrastructure of *APISENSE* as an SCA distributed system (cf. Figure 1). The *Service Component Architecture* (SCA)[6] standard is a set of specifications for building distributed application based on *Service-Oriented Architectures* (SOA) and *Component-Based Software Engineering* (CBSE) principles.

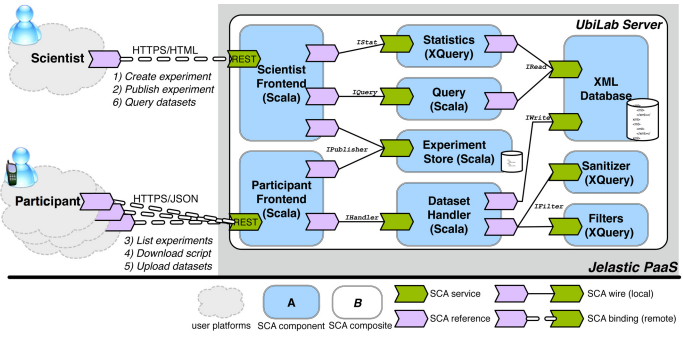

Fig. 1. Architecture of the *APISENSE* Web Infrastructure

All the components building the server-side infrastructure of *APISENSE* are hosted by a Cloud computing infrastructure [17]. The *Scientist Frontend* and *Participant Frontend* components are the endpoints for the two categories of users involved in the platform. Both components define all the services that can be remotely invoked by the scientists or the participants. For example, once authenticated, the scientist can create new experiments, follow their progression, and exploit the collected dataset directly from this web interface.

[6] `http://www.osoa.org`

Crowdsensing Library. To reduce the *learning curve*, we decided to adopt standard scripting languages in order to ease the description of experiments by the scientists. We therefore propose the *APISENSE* crowdsensing library as an extension of the JavaScript, CoffeeScript, and Python languages, which provides an efficient mean to describe an experiment without any specific knowledge of mobile device programming technologies (*e.g.*, Android SDK). The choice of these host languages was mainly motivated by their native support for JSON (*JavaScript Object Notation*), which is a lightweight data-interchange format reducing the communication overhead.

The *APISENSE* crowdsensing library adopts a reactive programming model based on the enlistment of handlers, which are triggered upon the occurence of specific events (cf. Section 3). In addition to that, the API of *APISENSE* defines a set of sensing functions, which can be used to retrieve specific contextual data form sensors. The library supports a wide range of features to build dataset from built-in sensors proposed by smartphones technologies, such as GPS, compass, accelerometers, bluetooth, phone call, application status (installed, running) in the context of *opportunistic* crowdsensing, but also to specify participatory sensing experiments (*e.g.*, surveys).

Privacy Filters. In addition to this script, the scientist can configure some privacy filters to limit the volume of collected data and enforce the privacy of the participants. In particular, *APISENSE* currently supports two types of filters. **Area filter** allows the scientist to specify a geographic area where the data requires to be collected. For example, this area can be the place where the scientist is interested in collecting a GSM signal (*e.g.*, campus area). This filter guarantees the participants that no data is collected and sent to the *APISENSE* server outside of this area. **Period filter** allows the scientist to define a time period during which the experiment should be active and collect data. For example, this period can be specified as the working hours in order to automatically discard data collected during the night, while the participant is expected to be at home.

By combining these filters, the scientist preserves the privacy of participants, reduces the volume of collected data, and improves the energy efficiency of the mobile application (cf. Section 4).

Deployment Model. Once an experiment is defined with the crowdsensing library, the scientist can publish it into the *Experiment Store* component in order to make it available to participants. Once published, two deployment strategies can be considered for deploying experiments. The former, called pull-based approach, is a proactive deployment strategy where participants download the list of experiments from the remote server. The latter, known as push-based approach, propagates the experiments list updates to the mobiles devices of participants. In the case of *APISENSE*, the push-based strategy would induce a communication and energy overhead and, in order to leave the choice to participants to select the experiments they are interested in, we adopted the pull-based approach as a deployment strategy. Therefore, when the mobile device of a participant connects to the *Experiment Store*, it sends its characteristics (including hardware, current location, sensor available and sensors that participants want to share) and receives the list of experiments that are compatible with the profile of the participant. The scientists can therefore configure the *Experiment Store* to limit the visibility of their experiments according the characteristics of participants. In order to reduce the

privacy risk, the device characteristics sent by the participants are not stored by the infrastructure and scientist cannot access to this information.

Additionally, the *Experiment Store* component is also used to update the behavior of the experiment once deployed in the wild. When an opportunistic connection is established between the mobile device and the *APISENSE* server, the version of the experiment deployed in the mobile device is compared to the latest version published in the server. The installed crowdsensing experiment is automatically updated with the latest version of the experiment without imposing participants to re-download manually the experiment. In order to avoid any versioning problem, each dataset uploaded automatically includes the version of the experiment used to build the dataset. Thus, scientists can configure the *Experiment Store* component in order to keep or discard datasets collected by older versions of the experiment.

2.2 Client-Side Library

Although our solution could be extended to other *Operating Systems*, the *APISENSE* mobile application is currently based on the Android operating system for the following reasons. First, the Android operating system is popular and largely adopted by the population, unit sales for Android OS smartphones were ranked first among all smartphone OS handsets sold worldwide during 2011 with a market share of 50.9% according to Gartner. Secondly, Android is an open platform supporting all the requirements for continuous sensing applications (*e.g.*, multitasking, background processing and ability to develop an application with continuous access to all the sensors), while for example, iOS 6 does no permit a continuous accelerometer sampling.

A participant willing to be involved in one or more crowdsensing experiments proposed by a scientist can download the *APISENSE* mobile application by flashing the QR code published on *APISENSE* website, install it, and quickly create an account on the remote server infrastructure. Once registered, the HTTP communications between the mobile device of the participant and the remote server infrastructure are authenticated and encrypted in order to reduce potential privacy leaks when transferring the collected datasets to the *APISENSE* server. From there, the participant can connect to the *Experiment Store*, download and execute one or several crowdsensing experiments proposed by scientists.

Figure 2 depicts the *APISENSE* software architecture. Building on the top of Android SDK, this architecture is mainly composed of four main parts allowing *i*) to interpret experiment scripts (*Facades*, *Scripting engine*) *ii*) to establish connection with the remote server infrastructure (*Network Manager*), *iii*) to control the privacy parameters of the user (*Privacy Manager*), and *iv*) to control power saving strategies (*Battery Manager*).

Scripting Engine. *Sensor Facades* bridge the Android SDK with the *Scripting Engine*, which integrates scripting engines based on the JSR 223 specification. We build a middleware layer *Bee.sense Scripting*, which exposes the sensors that can be accessed from the experiment scripts. This layer covers three roles: a *security role* to prevent malicious calls of critical code for the mobile device, a *efficiency role* by including a cache mechanism to limit system calls and preserve the battery, and an *accessibility role* to leverage the development of crowdsensing experiments, as illustrated in Section 3.

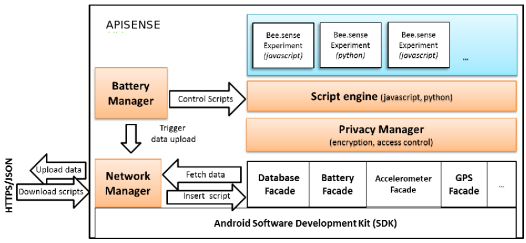

Fig. 2. Architecture of the *APISENSE* Mobile Application

Battery Manager. Although the latest generations of smartphones provides very powerful computing capabilities, the major obstacle to enable continuous sensing application refers to their energy restrictions. Therefore, in order to reduce the communication overhead with the remote server, which tends to be energy consuming, datasets are uploaded only when the mobile phone is charging. In particular, the battery manager component monitors the battery state and triggers the network manager component when the battery starts charging in order to send all the collected datasets to the remote server. Additionally, this component monitors the current battery level and suspends the scripting engine component when the battery level goes below a specific threshold (20% by default) in order to stop all running experiments. This threshold can be configured by the participant to decide the critical level of battery she wants to preserve to keep using her smartphone.

Privacy Manager. In order to cope with the ethical issues related to crowdsensing activities, the *APISENSE* mobile application allows *participants* to adjust their privacy preferences in order to constrain the conditions under which the experiments can collect data. As depicted in Figure 3, three categories of privacy rules are currently defined. Rules related to *location* and *time* specify geographical zone and time intervals conditions under which experiments are authorized to collect data, respectively. All the privacy rules defined by the participant are interpreted by the *Privacy Manager* component, which suspends the scripting engine component when one these rules is triggered. The last category of privacy rules refers to *authorization rules*, which prevent sensors activation or access to raw sensor data if the participant does not want to share this information. Additionally, a built-in component uses cryptography hashing to prevent experiments from collecting sensitive raw data, such as phone numbers, SMS text, or address book.

3 Crowdsensing Experiments

This section reports on four experiments that have been deployed in the wild using our platform. These examples demonstrate the variety of crowdsensing experimentations that are covered by the *APISENSE* infrastructre.

3.1 Revealing Users' Identity from Mobility Traces

This first experiment aimed at identifying the potential privacy leaks related to the sporadic disclosure of user's locations. To support this experiment, we developed a

APISENSE script, which reports every hour the location of a participant, known as Alice. Listing 1.1 describes the Python script we used to realize this experiment. This script subscribes to the periodic scheduler provided by the `time` facade in order to trigger the associated lambda function every hour. This function dumps a timestamped longitude/latitude position of Alice, which is automatically forwarded to the server.

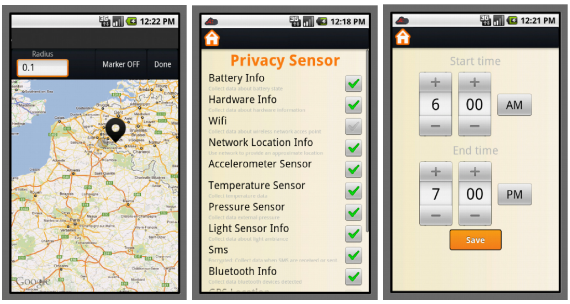

Fig. 3. Participant Privacy Preferences

```
time.schedule({'period': '1h'},
  lambda t: trace.add({ 'time': t.timestamp,
                        'lon': gps.longitude(), 'lat': gps.latitude() }))
```

Listing 1.1. Identifying GeoPrivacy Leaks (Python)

While this periodic report can be considered as anonymous since no identifier is included in the dataset, this study has shown that the identity of Alice can be semi-automatically be inferred from her mobility traces. To do so, we built a mobility model from the dataset we collected in order to identify clusters of Alice's locations as her *points of interest* (POI). By analyzing the size of the clusters and their relative timestamps, we can guess that the largest POI in the night relates to the house of Alice. Invoking a geocoding service with the center of this POI provides us a postal address, which can be used as an input to the yellow pages in order to retrieve a list of candidate names. In parallel, we can identify the places associated to the other POIs by using the Foursquare API, which provides a list of potential places where Alice is used to go. From there, we evaluate the results of search queries made on Google by combining candidate names and places and we rank the names based on the number of pertinent results obtained for each name. This heuristic has demonstrated that the identity of a large population of participants can be easily revealed by sporadically monitoring her location [11].

3.2 Building WiFi/GSM Signal Open Data Maps

This second experiment illustrates the benefits of using *APISENSE* to automatically build two open data maps from datasets collected in the wild. Listing 1.2 is a JavaScript script, which is triggered whenever the location of a participant changes by a distance of 10 meters in a period of 5 minutes. When these conditions are met, the script builds a trace which contains the location of the participant and attaches WiFi and GSM networks characteristics.

```
trace.setHeader('gsm_operator', gsm.operator());
location.onLocationChanged({ period: '5min',
 distance: '10m' }, function(loc) {
   return trace.add({
     time: loc.timestamp,
     lat: loc.latitude, lon: loc.longitude,
     wifi: { network_id: wifi.bssid(),
           signal_strength: wifi.rssi() },
     gsm: { cell_id: gsm.cellId(),
          signal_strength: gsm.dbm() } });
});
```

Listing 1.2. Building an Open Data Map (JavaScript)

From the dataset, collected by three participants over one week, we build an QuadTree geospatial index to identify the minimum bounding rectangles that contain at least a given number of signal measures. These rectangles are then automatically colored based on the median signal value observed in this rectangle (cf. Figure 4). This map has been assessed by comparing it with a ground truth map locating the GSM antennas and WiFi routers[7].

3.3 Detecting Exceptions Raised by User Applications

The third experiment highlights that *APISENSE* does not impose to collect geolocated dataset and can also be used to build realistic dataset focusing on the exceptions that are raised by the participants' applications. To build such a dataset, Listing 1.3 describes a CoffeeScript script that uses the Android logging system (`logCat`) and subscribes to error logs (`'*:E'`). Whenever, the reported log refers to an exception, the script builds a new trace that contains the details of the log and retrieves the name of the application reporting this exception.

```
logcat.onLog {filter: '*:E'},
  (log) -> if log.message contains 'Exception'
    trace.save
      message: log.message,
      time: log.timestamp,
      application: apps.process(log.pid).applicationName,
      topTask: apps.topTask().applicationName
```

Listing 1.3. Catching Mobile Applications' Exceptions (CoffeeScript)

Once deployed in the wild, the exceptions reported by the participants can be used to build a taxonomy of exceptions raised by mobile applications. The Figure 5, depicts the results of this experiment based on a dataset collected from three participants over one month. In particular, one can observe that a large majority of errors reported by the participant's applications are related to permission or database accesses, which can usually be fixed by checking that the application is granted an access prior to any invocation of a sensor or the database. This experiment is a preliminary step in order to better identify bugs raised by applications once they are deployed in the wild as we believe that the diversity of mobile devices and operating conditions makes difficult the application of traditional *in vitro* testing techniques.

[7] `http://www.cartoradio.fr`

Fig. 4. GSM Open Data Map

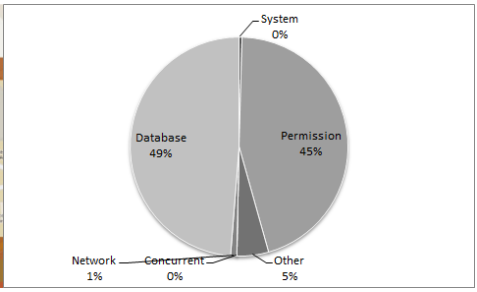

Fig. 5. Exceptions Taxonomy

3.4 Experimenting Machine Learning Models

The fourth experiment does not only collect user-contributed datasets, but also deals with the empirical validation of models on a population of participants. In this scenario, the scientist wanted to assess the machine learning model she defined for detecting the activity of the users: *walking*, *sitting*, *standing*, *running*, or *climbing and down stairs*. To assess this model, she deployed a script that integrates two phases: an exploration phase and an exploitation one. To set up this experiment, we extended the scripting library by integrating a popular machine learning [14] and adding a new facade to use its features from script. The script (cf. Listing 1.4) therefore starts with an exploration phase in order to learn a specific user model. During this phase, *APISENSE* generates some dialogs to interact with the participant and ask her to repeat some specific movements. The script automatically switches to the next movement when the model has recorded enough raw data from the accelerometer to provide an accurate estimation. Once the model is considered as complete, the script dynamically replace the timer handler to switch into the exploration phase. The dataset collected by the server-side infrastructure of *APISENSE* contain the model statisitcs observed for each participant contributing to the experiment.

Figure 6 reports on the collected statistics of this experiment and shows that the prediction model developed by the scientist matches quite accurately the targeted classes.

	Predicted class						**Acc** (%)
	Walk	Jog	Stand	Sit	Up	Down	
Walk	**66**	0	4	0	0	0	94,3
Jog	0	**21**	0	0	0	0	100
Stand	4	0	**40**	0	0	0	90,9
Sit	0	0	2	**83**	0	0	97,6
Up stair	0	0	0	0	**22**	0	100
Down stair	0	0	0	0	0	**11**	100

Fig. 6. Representative Confusion Matrix

```
var classes = ["walk","jog","stand", "sit", "up", "down"];
var current = 0; var buffer = new Array();
var model = weka.newModel(["avrX","avrY",...], classes);
var filter = "|(dx>"+delta+")(dy>"+delta+")(dz>"+delta+")";

var input = accelerometer.onChange(filter,
   function(acc) { buffer.push(acc) });

var learn = time.schedule({ period: '5s' }, function(t) {
  if (model.learn(classes[current]) >= threshold) {
    current++;
  }
  if (current < classes.length) { // Learning phase
    input.suspend();
    var output = dialog.display({ message: "Select movement", spinner: classes });
    model.record(attributes(buffer), output);
    sleep('2s');
    buffer = new Array();
    input.resume();
  } else { // Exploitation phase
    dialog.display({message: "Learning phase completed"});
    learn.cancel();
    model.setClassifier(weka.NAIVE_BAYES);
    time.schedule({ period: '5s' }, function(t) {
      trace.add({
        position: model.evaluate(attributes(buffer)),
          stats: model.statistics() });
      buffer = new Array();
} } });
```

Listing 1.4. Assessing Machine Learning Models (JavaScript)

4 Empirical Validations

Evaluating the Programming Models. In this section, we compare the *APISENSE* crowdsensing library to two state-of-the-art approaches: AnonySense [20] and Pogo [3]. We use the *RogueFinder* case study, which has been introduced by AnonySense and recently evaluated by Pogo. Listings 1.5 and 1.6 therefore reports on the implementation of this case study in AnonySense and Pogo, as decribed in the literature, while Listing 1.7 describes the implementation of this case study in *APISENSE*.

```
(Task 25043) (Expires 1196728453)
(Accept (= @carrier 'professor'))
(Report (location SSIDs) (Every 1 Minute)
(In location
  (Polygon (Point 1 1) (Point 2 2)
  (Point 3 0))))
```

Listing 1.5. Implementing *RogueFinder* in AnonySense

One can observe that *APISENSE* provides a more concise notation to describe crowdsensing experiments than the state-of-the-art approaches. This concisition is partly due to the fact that *APISENSE* encourages the separation of concerns by externalizing the management of time and space filters in the configuration of the experiment. A direct impact of this property is that the execution of *APISENSE* scripts better preserves the battery of the mobile device compared to Pogo, as it does not keep triggering the script when the user leaves the assigned polygon. Nonetheless, this statement is only based on an observation of Pogo as the library is not made freely available to confirm this empirically.

```
function start() {
  var polygon = [{x:1, y:1}, {x:2, y:2}, {x:3, y:0}];
  var subscription = subscribe('wifi-scan', function(msg){
    publish(msg, 'filtered-scans');
  }, { interval: 60 * 1000 });
  subscription.release();
  subscribe('location', function(msg) {
    if (locationInPolygon(msg, polygon))
      subscription.renew();
    else
      subscription.release();
  });
}
```

Listing 1.6. Implementing *RogueFinder* in POGO (JavaScript)

```
time.schedule { period: '1min' },
  (t) -> trace.add { location: wifi.bssid() }
```

Listing 1.7. Implementing *RogueFinder* in *APISENSE* (CoffeeScript)

Evaluating the Energy Consumption. In this section, we compare the energy consumption of *APISENSE* to a native Android application and another state-of-the-art crowdsensing solution: FUNF [1]. FUNF provides an Android toolkit to build custom crowdsensing applications à la carte. For each technology, we developed a sensing application, which collects the battery level every 10 minutes. We observed the energy consumption of these applications and we report their consumption in Figure 7.

Compared to the baseline, which corresponds to the native Android application, one can observe that the overhead induced by our solution is lower than the one imposed by the FUNF toolkit. This efficiency can be explained by the various optimizations included in our crowdsensing library. Although more energyvorous than a native application, our solution does not require advanced skills of the Android development framework and covers the deployment and reporting phases on behalf of the developer.

As the energy consumption strongly depends on *i)* the nature of the experiment, *ii)* the types of sensors accessed, and *iii)* the volume of produced data, we conducted a second experiment in order to quantify the impact of sensors (cf. Figure 8). For this experiment, we developed three scripts, which we deployed separately. The first script, labelled Bee.sense + Bluetooth, triggers a Bluetooth scan every minute and collects both the battery level as well as the resulting Bluetooth scan. The second script, Bee.sense + GPS, records every minute the current location collected from the GPS sensor, while the third script, Bee.sense + WiFi, collects a WiFi scan every minute. These experiments demonstrate that, even when stressing sensors, it is still possible to collect data during a working day without charging the mobile phone (40% of battery left after 10 hours of pulling the GPS sensor).

5 Related Work

A limited number of data collection tools are freely available on the market. SYSTEMSENS [8], a system based on Android, focuses on collecting usage context (*e.g.*, CPU, memory, network info, battery) of smartphones in order to better understand the battery consumption of installed applications. Similarly, LIVELABS [19] is a tool to measure

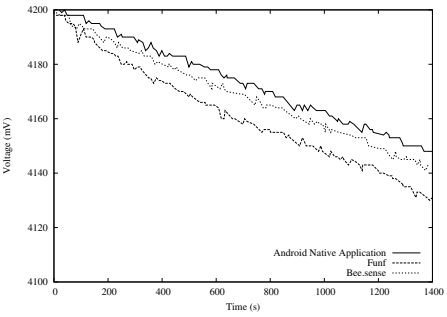

Fig. 7. Energy Consumptions of Android, *APISENSE*, and FUNF

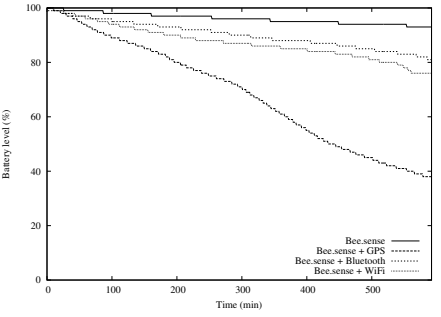

Fig. 8. Impact of *APISENSE* on the Battery Lifespan

wireless networks in the field with the principal objective to generate a complete network coverage map in order to help client to select network interface or network operators to identify blind spots in the network. However, all these tools are closed solutions, designed for collecting specific datasets and cannot be reused in unforeseen contexts in contrast to *APISENSE*. Furthermore, these projects are typical experiments deployed on mobile devices, without providing any privacy guarantee.

FUNF [1] is an Android toolkit focusing on the development of sensing applications. FUNF in a box is a service provided by FUNF to easily build a dedicated sensing application from a web interface, while data is periodically published via the Dropbox service. As demonstrated in Section 4, the current version of FUNF does not provide any support for saving energy nor preserving user privacy. Furthermore, the current solution does not support the dynamic re-deployment of experiments once deployed in the wild.

More interestingly, MYEXPERIENCE [9] is a system proposed for Windows mobile smartphones, tackling the *learning curve* issue by providing a lightweight configuration language based on XML in order to control the features of the application without writing C# code. MYEXPERIENCE collects data using a *participatory* approach—*i.e.*, by interacting with users when a specific event occurs (*e.g.*, asking to report on the quality of the conversation after a phone call ends). However, MYEXPERIENCE does not consider severals critical issues, such as maintaining the privacy of participants or the strategic deployment of experiments. Even if an experiment can be modified in the wild, each experiment still requires a physical access to the mobile device in order to be installed, thus making it difficult to be applied on a large population of participants.

In the literature, severals deployment of crowdsensing applications strategies have been studied. For example, ANONYSENSE [20] uses—as *APISENSE*—a pull-based approach where mobile nodes periodically download all sensing experiments available on the server. A crowdsensing experiment is written in a domain-specific language and defines when a mobile node should sense and under which conditions the report should be submitted to the server. However, ANONYSENSE does not provide any mechanism to filter the mobile nodes able to download sensing experiments, thus introducing a communication overhead if the node does not match the experiment requirements.

On the contrary, PRISM [7] and POGO [3] adopts a push-based approach to deploy sensing experiments over mobile nodes. PRISM is a mobile platform, running on

Microsoft Windows Mobile 5.0, and supporting the execution of generic *binary code* in a secure way to develop real-time participatory sensing applications. To support real-time sensing, PRISM server needs to keep track of each mobile node and the report they periodically send (*e.g.*, current location, battery left) before selecting the appropriate mobile phones to push application binaries. POGO proposes a middleware for building crowdsensing applications and using the XMPP protocol to disseminate the datasets. Nonetheless, POGO does not implement any client-side optimizations to save the mobile device battery (*e.g.*, area and period filters) as it systematically forwards the collected data to the server.

SensorSafe [5] is another participatory platform, which allows users to share data with privacy guaranties. As our platform, *SensorSafe* provides fine-grained temporal and location access control mechanisms to keep the control of data collected by sensors on mobile phone. However, participants have to define their privacy rules from a web interface while in *APISENSE* these rules are defined directly from the mobile phone.

6 Conclusion

While it has been generally acknowledged as a keystone for the mobile computing community, the development of crowdsensing platforms remains a sensitive and critical task, which requires to take into account a variety of requirements covering both technical and ethical issues.

To address these challenges, we report in this paper on the design and the implementation of the *APISENSE* distributed platform. This platform distinguishes between two roles: *scientists* requiring a sustainable environment to deploy sensing experiments and *participants* using their own mobile device to contribute to scientific experiments. On the server-side, *APISENSE* is built on the principles of Cloud computing and offers to scientists a modular service-oriented architecture, which can be customized upon their requirements. On the client-side, the *APISENSE* platform provides a mobile application allowing to download experiments, executing them in a dedicated sandbox and uploading datasets to the *APISENSE* server. Based on the principle of *only collect what you need*, the *APISENSE* platform delivers an efficient yet flexible solution to ease the retrieval of realistic datasets.

References

1. Aharony, N., Pan, W., Ip, C., Khayal, I., Pentland, A.: Social fmri: Investigating and shaping social mechanisms in the real world. In: Pervasive and Mobile Computing (2011)
2. Biagioni, J., Gerlich, T., Merrifield, T., Eriksson, J.: EasyTracker: automatic transit tracking, mapping, and arrival time prediction using smartphones. In: 9th Int. Conf. on EmbeddedNetworked Sensor Systems. ACM (November 2011)
3. Brouwers, N., Langendoen, K.: Pogo, a Middleware for Mobile Phone Sensing. In: Narasimhan, P., Triantafillou, P. (eds.) Middleware 2012. LNCS, vol. 7662, pp. 21–40. Springer, Heidelberg (2012)
4. Burke, J.A., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., Srivastava, M.B.: Participatory Sensing (2006)

5. Choi, H., Chakraborty, S., Greenblatt, M., Charbiwala, Z.M., Srivastava, M.B.: Sensorsafe: Managing health-related sensory information with fine-grained privacy controls. Technical report (TR-UCLA-NESL-201009-01) (September 2010)
6. Cuff, D., Hansen, M., Kang, J.: Urban Sensing: Out of the Woods. Communications of the ACM 51(3) (2008)
7. Das, T., Mohan, P., Padmanabhan, V.N., Ramjee, R., Sharma, A.: Prism: Platform for Remote Sensing Using Smartphones. In: 8th Int. Conf. on Mobile Systems, Applications, and Services. ACM (2010)
8. Falaki, H., Mahajan, R., Estrin, D.: SystemSens: a tool for monitoring usage in smartphone research deployments. In: 6th ACM Int. Work on Mobility in the Evolving Internet Architecture (2011)
9. Froehlich, J., Chen, M.Y., Consolvo, S., Harrison, B., Landay, J.A.: Myexperience: a system for in situ tracing and capturing of user feedback on mobile phones. In: 5th Int. Conf. on Mobile Systems, Applications, and Services. ACM (2007)
10. Haderer, N., Rouvoy, R., Seinturier, L.: A preliminary investigation of user incentives to leverage crowdsensing activities. In: 2nd International IEEE PerCom Workshop on Hot Topics in Pervasive Computing (PerHot). IEEE (2013)
11. Killijian, M.-O., Roy, M., Trédan, G.: Beyond San Fancisco Cabs: Building a *-lity Mining Dataset. In: Work. on the Analysis of Mobile Phone Networks (2010)
12. Lane, N.D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., Campbell, A.T.: A Survey of Mobile Phone Sensing. IEEE Communications Magazine 48(9) (2010)
13. Liu, L., Andris, C., Biderman, A., Ratti, C.: Uncovering Taxi Driver's Mobility Intelligence through His Trace. In: IEEE Pervasive Computing (2009)
14. Liu, P., Chen, Y., Tang, W., Yue, Q.: Mobile weka as data mining tool on android. In: Advances in Electrical Engineering and Automation, pp. 75–80 (2012)
15. Miluzzo, E., Lane, N.D., Lu, H., Campbell, A.T.: Research in the App Store Era: Experiences from the CenceMe App Deployment on the iPhone. In: 1st Int. Work. Research in the Large: Using App Stores, Markets, and Other Wide Distribution Channels in UbiComp Research (2010)
16. Mun, M., Reddy, S., Shilton, K., Yau, N., Burke, J., Estrin, D., Hansen, M., Howard, E., West, R., Boda, P.: PEIR, The Personal Environmental Impact Report, as a Platform for Participatory Sensing Systems Research. In: 7th Int. Conf. on Mobile Systems, Applications, and Services. ACM (2009)
17. Paraiso, F., Haderer, N., Merle, P., Rouvoy, R., Seinturier, L.: A Federated Multi-Cloud PaaS Infrastructure. In: 5th IEEE Int. Conf. on Cloud Computing (2012)
18. Roy, M., Killijian, M.-O.: Brief Announcement: A Platform for Experimenting with Mobile Algorithms in a Laboratory. In: 28th Annual ACM Symp. on Principles of Distributed Computing, ACM (2009)
19. Shepard, C., Rahmati, A., Tossell, C., Zhong, L., Kortum, P.: LiveLab: measuring wireless networks and smartphone users in the field. ACM SIGMETRICS Performance Evaluation Review 38(3) (2011)
20. Shin, M., Cornelius, C., Peebles, D., Kapadia, A., Kotz, D., Triandopoulos, N.: AnonySense: A System for Anonymous Opportunistic Sensing. In: Pervasive and Mobile Computing (2010)
21. Sohn, T., et al.: Mobility Detection Using Everyday GSM Traces. In: Dourish, P., Friday, A. (eds.) UbiComp 2006. LNCS, vol. 4206, pp. 212–224. Springer, Heidelberg (2006)