Noname manuscript No.
(will be inserted by the editor)

Managing Big Data Experiments on Smartphones

Georgios Larkou · Marios Mintzis · Panayiotis G. Andreou · Andreas Konstantinidis · Demetrios Zeinalipour-Yazti

Received: date / Accepted: date

Abstract The explosive number of smartphones with ever growing sensing and computing capabilities have brought a paradigm shift to many traditional domains of the computing field. Re-programming smartphones and instrumenting them for application testing and data gathering at scale is currently a tedious and time-consuming process that poses significant logistical challenges. Next generation smartphone applications are expected to be much larger-scale and complex, demanding that these undergo evaluation and testing under different real-world datasets, devices and conditions. In this paper, we present an architecture for managing such large-scale data management experiments on real smartphones. We particularly present the building blocks of our architecture that encompassed smartphone sensor data collected by the crowd and organized in our big data repository. The given datasets can then be replayed on our testbed comprising of real and simulated smartphones accessible to developers through a web-based interface. We present the applicability of our architecture through a case study that involves the evaluation of individual components that are part of a complex Indoor Positioning System for smartphones, coined Anyplace, which we have developed over the years. The given study shows how our architecture allows us to derive novel insights into the performance of our algorithms and applications, by simplifying the management of large-scale data on smartphones.

Keywords Experimental Testbed, Big Data, Sensor Mockups, Smartphones

Department of Computer Science, University of Cyprus,
1 University Avenue, P.O. Box 20537,
1678 Nicosia, CYPRUS.
Tel.: +357-22-892755
Fax.: +357-22-892701
E-mail: {glarkou,mmintz01,panic,akonstan,dzeina}@cs.ucy.ac.cy

1 Introduction

The continuous improvements of smartphone devices and embedded sensor systems during the past decade have enabled researchers to explore complex interdisciplinary areas (e.g., behavioral sciences, social sciences) from the big data perspective. This facilitates understanding of the physical world at an extremely high fidelity and interpretation of real-life problems by analyzing individual behavior through mobility, communication and interaction patterns. The latter patterns can be obtained from modern smartphone devices that continuously provide new and more efficient means for big data generation through their enhanced computing and multi-sensing capabilities (e.g., geolocation, proximity, ambient light, accelerometer, camera, microphone, etc.).

Smartphone users are constantly moving and sensing thus generating large amounts of data contributing to the evolution of new services and applications [1], also known as crowdsourcing, which is gradually becoming the prevalent mean of data gathering but also processing such vast amounts of information from large groups of people. Typical examples of crowdsourcing applications include commercial services such as Gwap.net's ESP image tagging game, reCAPTCHA.net's book correction service, and specialized marketplaces for assigning crowdsourcing tasks (e.g., Amazon's Mechanical Turk) and academic techniques in data management [2, 3], network management [4, 5], source-code development [6], computational linguistics [7] and active learning [8]. These applications often tend to share information acquired from onboard sensor devices (e.g., camera, GPS, etc.) resulting in an explosion of geo-tagged temporal data that is communicated over the internet.

Re-programming smartphones and instrumenting them for application testing and data gathering at scale is currently a tedious, time-consuming process that poses significant logistical challenges. Moreover, allowing seamless experimental repeatability and standardization is a challenging task for smartphone-oriented research. Looking at other research areas, somebody will realize that open benchmarking datasets and associated ground truth datasets have played an important role in academic and industrial research over the last decades. For instance, the TREC Conference series co-sponsored by National Institute of Standards and Technology (NIST) of the U.S. Commerce Department is heavily embarked by the information retrieval community. Similarly, the TPC (Transaction Processing Performance Council) non-profit corporation, founded to define transaction processing and database benchmarks, is heavily embarked by the data management community.

To this end, we have implemented and demonstrated *SmartLab*[1] [9], a comprehensive architecture for managing a cluster of both *Android Real Devices (ARDs)* and *Android Virtual Devices (AVDs)*, which are managed via an intuitive web-based interface. Amongst others, our current architecture is ideal for conducting *fine-grained* and *low-level* control and experimentation of various algorithms over real heterogeneous smartphone devices, gathering and managing large-scale data contributed by the crowd (e.g., WiFi RSS data [10]) and storing them in big data infrastructures, as well as performing experiments that require the engagement of physical and mock-up sensors [11] (Figure 1).

[1] Available at *http://smartlab.cs.ucy.ac.cy/*

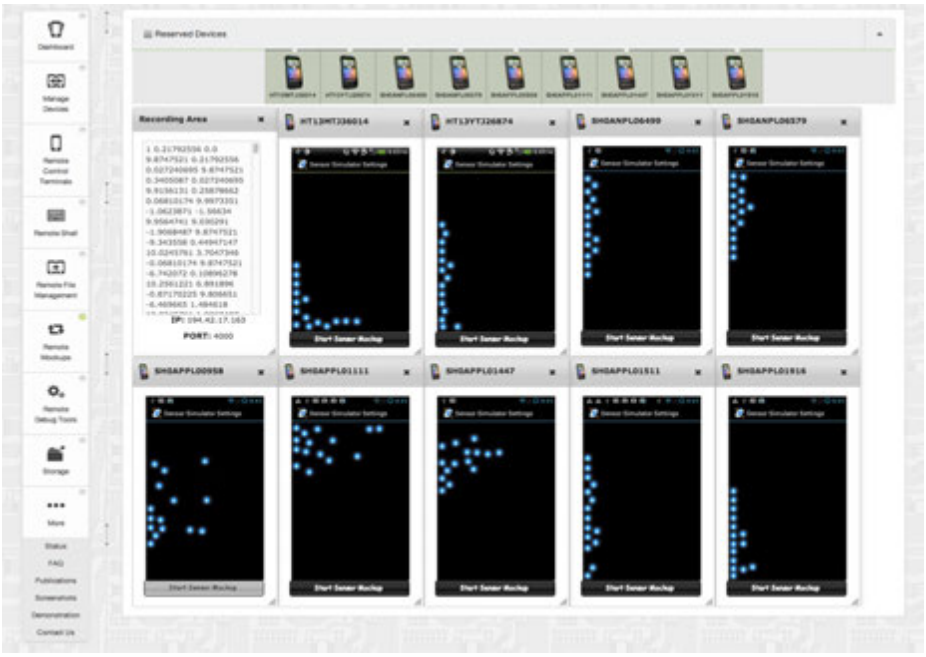


Fig. 1 An example mockup experiment in SmartLab [11], where a user feeds real smartphones with sensor readings from a big data store and overviews the results through a Web 2.0 User Interface.

SmartLab's current hardware consists of over 40 Android devices that are connected through a variety of means (i.e., *wired, wireless* and *virtual*) to our *private cloud (datacenter)*. Through an intuitive web-based interface, users can upload and install Android executables on a number of devices concurrently, capture their screen, transfer files, issue UNIX shell commands, "feed" the devices with GPS/sensor mockups and many other exciting features.

Looking at the latest trends, we observe that open smartphone OSs, like Android, are the foundation of emerging Personal Gadgets (PGs): eReaders (e.g., Barnes & Noble), Smartwatches (e.g., Motorola MOTOACTV), Rasberry PIs, SmartTVs and SmartHome appliances in general. SmartLab can be used to allow users manage all of their PGs at a fine-grain granularity (e.g., screen-capture, interactivity, filesystem). Additionally, we anticipate that the overtake of PC sales by Smartphone sales will soon also introduce the notion of Beowulf-like or Hadoop-like smartphone clusters for power-efficient computations and data analytics. Moreover, one might easily build powerful computing testbeds out of deprecated smartphones, like *Micro-cellstores* [12], as users tend to change their smartphones more frequently than their PC. Consequently, providing a readily available PG management middleware like SmartLab will be instrumental in facilitating these directions. Finally, SmartLab is a powerful tool for Internet service providers and other authorities that require to provide remote support for their customers as it can be used to remotely control and maintain these devices.

In this paper, we present the major components of the SmartLab architecture and how these can facilitate management of large-scale data experiments on

real smartphones. We particularly present the building blocks of our architecture that encompass smartphone sensor data collected by the crowd and organized in our big data repository. The given datasets can then be replayed on our testbed comprising of real and simulated smartphones accessible to developers through a web-based interface. We demonstrate the applicability of our architecture through experiments on a complex Indoor Positioning System, coined Anyplace[2][13], which we have developed over the years that highlights how SmartLab can process considerable amount of geospatial data and transparently manage the complexity of the aforementioned experimental issues. The performance of the proposed system is also evaluated with respect to Samsung's Remote Test Lab, PerfectoMobile testbed and the OpenIntents Sensor Simulator.

The contributions of this work are summarized as follows:

- We present the architecture behind SmartLab, an innovative smartphone programming cloud that enables fine-grained control over both ARDs and AVDs via an intuitive web-based interface.
- We present a detailed description of the SmartLab components that facilitate management of big data experiments; the big data and algorithms repositories and the Remote Mockup Library.
- We demonstrate how SmartLab is used to manage and process considerable amounts of geospatial data and consequently has facilitated the development and testing of Anyplace.
- We experimentally demonstrate the applicability of SmartLab focusing on three dimensions: i) Deployment; ii) Management/Monitoring; and iii) Experimentation using sensor mockup data. Additionally, we conduct an experiment that shows how SmartLab facilitates new research directions.

The rest of the paper is organized as follows: Section 2 overviews related work. Section 3 presents the SmartLab testbed including its architecture and major components, followed by Section 4 that provides background information related to indoor positioning systems and describes the Anyplace application. Next, Section 5 describes our experimental methodology and Section 6 presents our experimental evaluation. Finally, Section 7 concludes the paper and provides future directions.

2 Related Work

This section provides a concise overview of the related work. SmartLab has been inspired by *PlanetLab* [14] and *Emulab* [15], both of which have pioneered global research networks; *MoteLab* [16], which has pioneered sensor network research and *Amazon Elastic Compute Cloud (EC2)*. None of the aforementioned efforts focused on smartphones and thus those testbeds had fundamentally different architectures and desiderata. In the following subsections, we will overview testbeds that are related to SmartLab as well as big data and localization technologies.

[2] Anyplace, `http://anyplace.cs.ucy.ac.cy/`

2.1 Smartphone Testbeds

There are currently several commercial platforms providing remote access to real smartphones, including *Samsung's Remote Test Lab [17], PerfectoMobile [18], Device Anyware [19] and AT&T ARO [20].* These platforms differ from SmartLab in the following ways: i) they are mainly geared towards application testing scenarios on individual smartphones; and ii) they are *closed* and thus, neither provide any insights into how to efficiently build and run smartphone applications at scale nor support the wide range of functionality provided by SmartLab like sensors, mockups and automation.

Sandia National Laboratories has recently presented *MegaDroid [21]*, a 520-node PC cluster worth $500K that deploys 300,000 AVD simulators. MegaDroid's main objective is to allow researchers to massively simulate real users. Megadroid only focuses on AVDs while SmartLab focuses on both ARDs and AVDs as well as the entire management ecosystem, providing means for *fine-grained* and *low-level* interactions with real devices of the testbed as opposed to virtual ones.

2.2 People-centric Testbeds

There is another large category of systems that focuses on opportunistic and participatory smartphone sensing testbeds with real custodians, e.g., *PRISM [22]*, *CrowdLab [23]* and *PhoneLab [24]*, but those are generally complementary as they have different desiderata than SmartLab.

Let us for instance focus on PhoneLab, which is a participatory smartphone sensing testbed that comprises of students and faculty at the University of Buffalo. PhoneLab does not allow application developers to obtain screen access, transfer files or debug applications, but only enables programmers to initiate data logging tasks in an offline manner. PhoneLab is targeted towards data collection scenarios as opposed to fine-grained and low-level access scenarios we support in this work, like deployment and debugging. Additionally, PhoneLab is more restrictive as submitted jobs need to undergo an Institutional Review Board process, since deployed programs are executed on the devices of real custodians.

Finally, UC Berkeley's Carat project [25] provides collaborative energy diagnosis and recommendations for improving the smartphone battery life from more than half a million crowd-powered devices. SmartLab is complementary to the above studies as we provide insights and experimental results for a variety of modules that could be exploited by these systems.

2.3 Big Data

Big data refers to data sets whose size and structure strains the ability of commonly used relational DBMSs to capture, manage, and process the data within a tolerable elapsed time [26]. The *Volume-Velocity-Variety* of information in this kind of datasets give rise to the big data challenge, which is also known as the 3V challenge.

The *volume* of such datasets is in the order of few terabytes (TB) to petabytes (PB). Examples of such volumes are the U.S. Library of Congress that in April

2011 had more than 235 TB of data stored and the World of Warcraft online game using 1.3 PB of storage to maintain its game, the German Climate Computing Center (DKRZ) storing 60 PB of climate data. The *velocity* of information in social media applications (such as photovoltaic, traffic and other monitoring apps) can grow exponentially as users join the community. Such growth can produce unprecedented volumes of data streams. For example, Ontario's Meter Data Management and Repository (MDM/R) [27] stores, processes and manages data from 4.6 million smart meters in Ontario, Canada and provides hourly billing quantity and extensive reports counting 110 million meter reads per day on an annual basis that exceeds the number of debit card transactions processed in Canada. Furthermore, the *variety* of data can be anything from structured (relational or tabular) to semi-structured (XML or JSON) or even unstructured (Web text and log files) data and combination thereof. For example, Google's experimental robot cars [28], which have navigated thousands of miles of California roads, use an artificial-intelligence technique tackling big data challenges, parsing vast quantities of data and making decisions instantaneously.

Due to the high demand for big data management, the literature witnessed an emergence of new techniques and tools for taming big data. Currently, there are disk-resident approaches [29–31] founded on the popular Hadoop Map-Reduce[32] programming paradigm, as well as main-memory and stream-oriented processing approaches of big data, founded on frameworks such as Apache Storm, Apache Spark and Stratosphere.

2.4 Localization Technologies

Indoor positioning systems have recently received considerable attention, both because GPS is unavailable in indoor spaces and consumes considerable energy. In particular, Assisted GPS (A-GPS) is obviously ubiquitously available but has an expensive energy tag and is also negatively affected from the environment (e.g., cloudy days, forests, downtown areas, etc.) Besides GPS, the localization community [33] proposed numerous proprietary solutions including: *Infrared, Bluetooth, visual or acoustic analysis, RFID, Inertial Measurement Units, Ultra-Wide-Band, Sensor Networks, Wireless LANs, etc.*; including their combinations into hybrid systems [34]. Most of these technologies deliver a high level of positioning accuracy, however they often require the deployment and calibration of expensive equipment, such as custom transmitters and antennas, which are dedicated to positioning. This is time consuming and implies high installation costs, while the approaches we discuss operate off-the-shelf with conventional smartphones.

Moreover other well known indoor localization approaches (such as *Airplace* [10]) exploit *Received Signal Strength (RSS)* values extracted through passive scanning of the beacon packets transmitted by neighboring APs as part of the standard network functionality. A vector of RSS values from APs within a user's vicinity (a.k.a. RSS fingerprint) is collected a priori, associated with a pre-defined location and stored in a so-called Radiomap, which in turn is utilized by interested users to localize themselves.

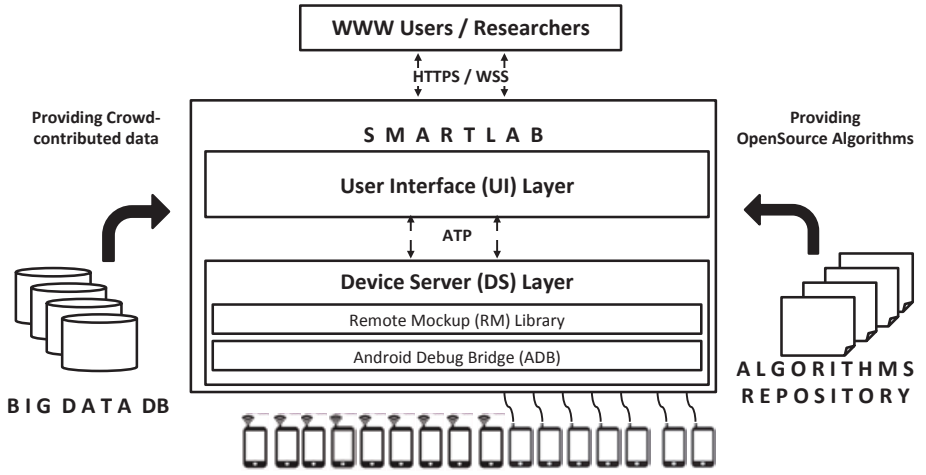


Fig. 2 The major components of the extended SmartLab Architecture involved in carrying out big data experiments.

2.5 Discussion

Smartlab provides an efficient and scalable way for managing and processing "Big Data" with mobile devices via a NoSQL infrastructure for facilitating the development and testing of mobile applications, which rely in high volume, high velocity and high variety of data. Note that to the best of our knowledge there are no other known approaches/platforms that allow smartphone applications to deal with such large amount of data.

In this paper, we demonstrate how SmartLab is used to manage and process considerable amounts of geospatial data and consequently has facilitated the development, testing and demonstration of Airplace and its successor project Anyplace considerably as explained later in Section 4. Firstly, we used SmartLab to collect *Received Signal Strength (RSS)* indicators from different WiFi chip-sets, which are important for RSS measurements, and smartphone sensor readings (from accelerometer, gyroscope and compass). Secondly, we utilized wirelessly connected smartphone devices in order to move around in a building localizing ourselves using Anyplace while exposing the smartphone screen on a remote web browser through SmartLab. The particular setting has proved considerably useful for demonstrations [35] at conferences as the bulk of existing AndroidScreenCapture software are both USB-based, which hinders mobility, but are also inefficient as they provide no compression or other optimizations. Thirdly, SmartLab allowed us to compare different indoor localization techniques over Anyplace on a variety of devices. Finally, SmartLab was used as a verification platform of data (e.g., RSS values) contributed by arbitrary users (i.e., the crowd).

Fig. 3 The SmartLab User Interface provides a set of tools that facilitates efficient and effective experimentation on smartphone devices.

3 SmartLab Architecture

SmartLab is an innovative Smartphone testbed that enables experimentation using both ARDs and AVDs via an intuitive web-based interface. In this section, we summarize the main architectural components of the SmartLab testbed and show how these can facilitate efficient and effective experiments that utilize different algorithms, big data repositories and heterogeneous devices. We start out by presenting a high level view of SmartLab's infrastructure (user interface, device server and hardware) and then move on to the specialized components that facilitate big data experiments; the Algorithms repository, the Remote File Management (RFM) Subsystem, the Big Data Repository and the Remote Mockup Library. SmartLab's architecture is illustrated in Figure 2.

3.1 User Interface Layer

SmartLab utilizes a clear, concise and usable interface (UI) that enables several modes of interaction with connected smartphone devices as illustrated in Figure 3. In particular, the UI supports: i) *Remote File Management (RFM)*, a terminal that allows users manage device files; ii) *Remote Control Terminals (RCT)*, a remote screen terminal that mimics smartphone touchscreen clicks and gestures but also enables users recording automation scripts for repetitive tasks; iii) *Remote Debug Tools (RDT)*, a debugging extension to the information available through the Android Debug Bridge (ADB); iv) *Remote Shells (RS)*, a shell enabling a wide variety of UNIX commands issued to the Android Linux kernels of allocated devices; v) *Remote Mockups (RM)*, a mockup subsystem for feeding ARDs and AVDs with GPS or sensor data traces encoded in XML for trace-driven experimentation.

The UI interactions are enabled either through *Websockets* for high-rate utilities or *AJAX-based communication* for low-rate utilities. In order to support web-

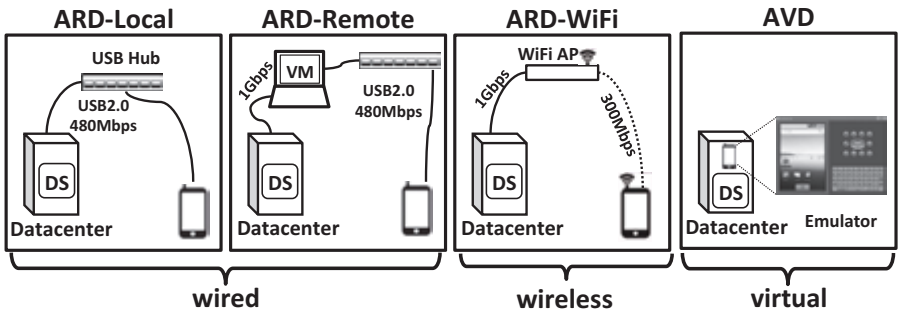


Fig. 4 Connection Modalities supported by SmartLab. ARD-Local: Android Real Device (ARD) mounted locally to the Device Server (DS) through USB; **ARD-Remote**: ARD mounted through a USB port on a gateway PC to DS through a wired network; **ARD-WiFi**: ARD connected to DS through a WiFi AP; and **AVD**: Android Virtual Device running on DS.

sockets on as many browsers as possible, we have modified the open kanaka websockify plugin[3]. The given plugin takes care of the initial Websocket handshake from within the browser but also shifts over to an SWF implementation (i.e., Adobe Flash), in cases where a browser is not HTML5-compliant, enabling truly-compliant cross-browser compatibility.

3.2 Device Server Layer

The *Device Server* (DS) layer enables the interaction of the UI with the smartphone devices. It allows administrator users to physically connect/disconnect smartphone devices so that they are available for usage via the UI. In order to support a larger number of connected devices with the current ADB release, we utilize multiple-DSs on each physical host of our datacenter each connecting 16 devices (ARDs or AVDs). This design choice is inspired from cloud environments and shared-nothing architectures deployed by big-data testbeds providing linear scalability by linearly engaging more resources. Additionally, the DS supports a number of tools required for maintenance purposes similarly to routers and printers.

3.3 Hardware Layer

SmartLab's hardware comprises of over 40 *Android Smartphones* and our *Datacenter*. Smartphones communicate with the DS via different communication modalities as illustrated in Figure 4. Most of the smartphone devices are connected to the server in ARD-Local mode, utilizing USB hubs. Similarly, a number of smartphones are also connected from within our research lab, in the same building, using the ARD-Remote mode. Additionally, a number of devices are also available in ARD-WiFi mode, utilizing the Departmental WiFi infrastructure, and others in ARD-Internet mode, utilizing 3G cards provided by a local Telecom operator.

[3] Kanaka, `https://github.com/kanaka/websockify/`

Fig. 5 Subset of the SmartLab smartphone fleet connected locally to our datacenter. More devices are connected over the wireless and wired network.

In the context of this work, ARD-Local and ARD-WiFi is extensively used. A connectivity experiment was conducted in order to evaluate the time-efficiency of the various connection modalities and is presented in Section 6.

Figure 5 presents a subset of the Smartlab smartphone fleet connected locally to our datacenter. The latter encompasses over 16TB of RAID-5 / SSD storage on an IBM X3550 as well as 320GB of main memory on 5 IBM / HP multiprocessor rackables. The majority of our smartphones came with pre-installed Android 2.1-2.3 (Eclair, Froyo, Gingerbread). Most of these devices were "rooted" (i.e., the process of obtaining root access) and upgraded to Android 4.0.4 (Ice Cream Sandwich) or later, using a custom XDA-Developers ROM, when their warranty expired. Notice that warranty and rooting are claimed to be irrelevant in Europe[4]. In SmartLab, rooted devices feature more functionality than non-rooted devices. Particularly, rooted devices in SmartLab can: i) mount remote filesystems over ssh; ii) provide a richer set of UNIX shell commands; and iii) support a higher performance to the screen capturing system by incorporating compression. Nevertheless, SmartLab has been designed from ground up for non-rooted devices, thus even without applying the rooting process will support all features other than those enumerated above.

3.4 Remote File Management (RFM) subsystem

In order to conduct experiments on smartphone devices, SmartLab utilizes the Remote File Management (RFM) subsystem that enables the deployment of ap-

[4] Free Software Foundation Europe, http://goo.gl/fZZQe

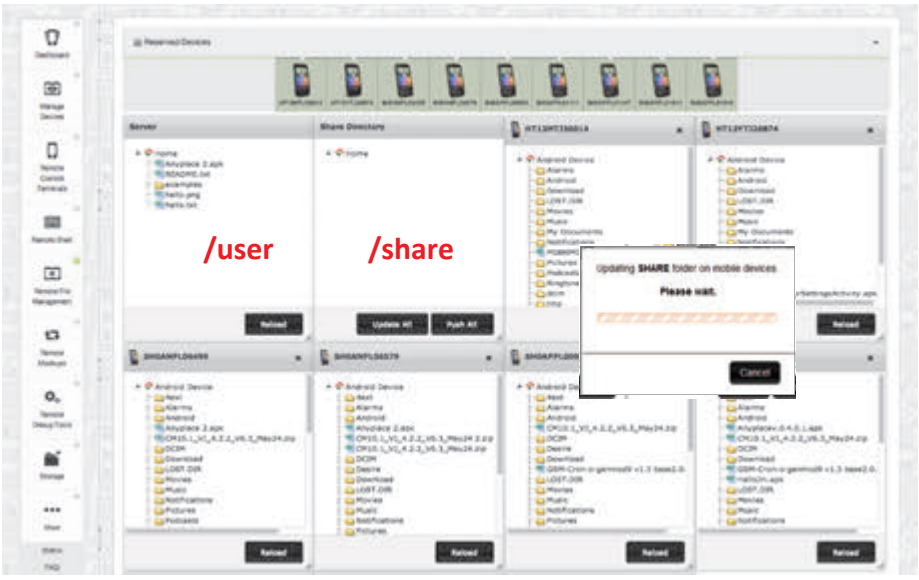


Fig. 6 Remote File Management (RFM) UI. A share folder enables to push/pull files to devices concurrently. FUSE-enabled devices can feature sshfs shares.

plications and/or related files on a number of devices simultaneously. Additionally, the RFM subsystem has been extensively used in the collection of experimental results stored as files on each device. In this subsection, we briefly describe the RFM UI and finally present some performance experiments for pushing a file and installing an application on a device using ADB pipelining.

We have constructed an intuitive HTML5/AJAX-based web interface for RFM, which enables the management of the local filesystems on smartphones *individually* but also *concurrently* (see Figure 6). In particular, our interface allows users to perform all common file management operations in a streamlined manner. The RFM interface starts by launching a separate window for each AVD or ARD that is selected by the user and displays a tree-based representation of its files and directories under the device's `/sdcard` directory. Similarly, it launches two additional frames (i.e., JQuery dialogs): i) one frame displays the users' "Home" directory (top-left); and ii) another frame displays a `/share` directory, which is illustrated in Figure 6 (top-center). The user is then able to move a single file or multiple files to multiple target devices. The Remote File Management subsystem is also responsible for replicating any files moved to the `/share` directory to each target device's `/sdcard/share` directory. Furthermore, an *Update All* button and a *Push All* button have been placed below the `/share` directory in order to support simultaneous updating or merging the `/share` directory on existing and newly reserved devices. In order to accomplish these operations, the RFM UI issues separate web requests, which include: i) the target device id (or multiple devices ids); ii) the absolute location of a single file (or multiple files); and iii) the type of operation. Requests are transmitted using AJAX, to the device server, which is responsible to execute the appropriate `adb push` and `adb pull` commands to

transfer files to or from a device, respectively, all over the *ATP* protocol discussed earlier.

3.5 Algorithms Repository

The Algorithm Repository stores a variety of open source and in-house developed algorithms (e.g., localization, crowdsourcing, p2p) for smartphone devices. The majority of these algorithms are packaged as stand-alone libraries (i.e., jar files) and can be used in the context of any experiment conducted using SmartLab. Documentation for each library is provided by its developer and located within the library file.

In the context of this paper, the Algorithm Repository provided the Indoor Positioning algorithms[5] utilized during the experimentation phase in Section 6. Since all indoor positioning algorithms evaluated during the experimentation phase required the same input (i.e., a vector of WiFi MAC addresses and the corresponding RSS measurements) and returned the same output (i.e., a geo-coordinate (longitude, latitude)), SmartLab facilitated easy integration of the data in a time-conserving deployment manner.

3.6 Big Data Repository

SmartLab employs a unified big data repository infrastructure in order to maintain multiple databases or files that can be utilized in experiments on smartphone devices. The repository employs mechanisms that promote easier experimentation both at the cluster-level as well as the smartphone-level. The big data repository architecture is illustrated in Figure 7.

Managing Big Data on the cluster: The data repository is currently located for distribution over a closed departmental network and has been used for storing data collected from research experiments (e.g., collecting own WiFi RSS data on campus [10], sensor data utilized by the Remote Mockup Library described next in Subsection 3.7).

Generally, the given store can be utilized to store billions of sensor readings stored in a document-oriented format that allow a researcher to test an algorithm or application using tens or hundreds of smartphone devices using automated scripts, similarly to [21] but with more extensive data traces. Since each sensor recording, depending on its duration, can store millions of modeled entries without any relations we had to consider using a NoSQL database, which provides simplicity of design, horizontal scaling and finer control over availability. We decided to use Couchbase 2.1.1 Community edition as our NoSQL database since it provides all aforementioned advantages over a relational database and it is able to accommodate unstructured JSON objects. Another advantage of Couchbase is the already built-in object-level cache, coined Memcache, which provides the ability to store and serve most frequent and recent queries immediately from the main memory (RAM) without the need of retrieving the results from the disk.

[5] Available under "Code" tab at http://dmsl.cs.ucy.ac.cy

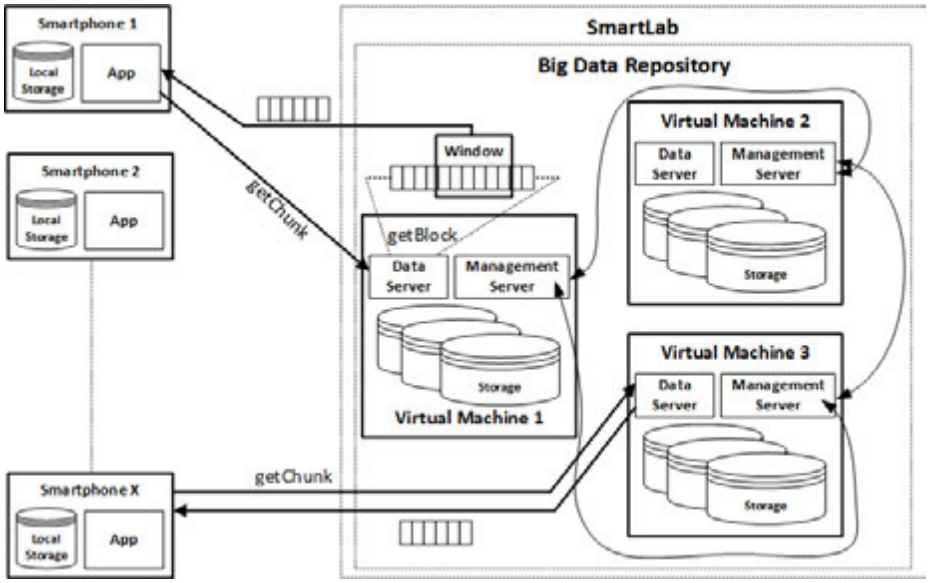


Fig. 7 Big Data Repository Deployment Architecture: SmartLab's big data repository is hosted by a coordinator couchbase management server virtual machine and two couchbase data servers virtual machines. It supports smartphone app query requests (`getChunk`) for data between two time instances via the `getBlock` view.

Our Couchbase setup utilizes a Couchbase bucket hosted by two Couchbase servers with 512MB of Memcache each (i.e., 1GB in total). The two servers can accommodate up to 6TBs of data on top of our infrastructure that encompasses over 16TB of RAID-5 / SSD storage. Additionally, if necessary, the experimental bucket could horizontally scale to tenths of virtual machines in order to provide enhanced availability. The data store uses 1206 bytes for storing one instance of 8 sensors readings in JSON format along with a Unix timestamp and a username associated to the SmartLab account of the user who initiated the recording.

Finally, Couchbase uses Views for processing the information stored in our database, allowing us to index and query our data. More specifically, a view creates an index on the stored information according to the format and structure defined in Javascript using a Map/Reduce paradigm. Moreover, Views can encapsulate specific key lookups, range queries, and aggregate lookups.

During the experimentation phase, we created two Views for retrieving sensor data sorted in our big data repository: coined `getBlock` and `userRecording`. The first one is used for retrieving sensor data for a specific recording of a specific user sorted based on unix timestamps stored in each JSON document. The latter is used for retrieving the sum of the available sensor data for a given recording, stored as JSON documents, which is used for calculating the size of the data that we should retrieve from the datastore for satisfying the calculated size of the "sliding window" that we are going to describe later.

Managing Big Data on Smartphones: Smartphones are not able to accommodate large amounts of data because of hardware/software limitations. For example, an Android application limits the amount of main memory an application can use between 16MB (e.g., HTC Desire) to 48 MB (e.g., Nexus 7). This is very limiting as nowadays smartphones can produce enormous amount of data each day; if a developer desires to log the measurements of 8 sensors for a single day every 100ms, this will produce approximately 993 MB of data (i.e., 10 readings/sec $\times$ 1206 Bytes $\times$ 60 sec $\times$ 60 min $\times$ 24 hours). However, the real problem lies in the ability to replay an existing log trace to another smartphone for experimental purposes.

Assume that we would like to perform an experiment that stores a sensor reading every millisecond and our Android device can only store 16 MB of data. This means that we can replay only about 14 seconds (13911ms) of the initial recording at once. Even if the experiments allow to sacrifice accuracy (e.g., record an instance every 100ms instead of every 1ms), this would again limit our playback time to 1400s (i.e., 23 minutes).

Furthermore, the authors in [36] show that by utilizing flash storage severely hampers application performance between 100% to 300% and in some extreme cases even by 2000%. More specifically, slower flash cards affect applications that are traditionally considered as CPU or network bound (e.g., web browser). Consequently, utilizing local storage for storing the aforementioned sensor readings and then retrieving them for application usage should be avoided, as it can lower overall application performance and consume more energy.

In order to overcome the aforementioned limitation and enable the repeatability of ideally "unlimited" sizes of recordings, we have developed a "sliding window" mechanism over the sensor data stream. This mechanism enables requesting only chunks of data from the datastore using the aforementioned `getBlock` view. During the experiments in Section 6, the server adjusted the window size for retrieving data according to the latency of the network between the Couchbase bucket and the device. The pre-calculated size (100MB) was used for requesting chunks of data from the datastore using the aforementioned getBlock view. Afterwards, the data was transferred to each pre-selected device for processing. When a device recognized that it reached processing of half the data (i.e., 50MB), then it automatically requested the next chunk of data from the server. This process repeated until the end of our recording.

3.7 Remote Mockup (RM) Library

A mockup provides part of a system's functionality enabling testing of a design. In the context of Android, Mockup refers to the process of extending an AVD's or ARD's particular sensor or GPS with custom values. Additionally, one important benefit of Mockups is that these can support the addition of sensors that may not exist in the hardware of a particular ARD (e.g., NFC, WiFi Direct, etc).

In order to support both GPS and other sensor mockups in SmartLab, (e.g., *accelerometer, compass, orientation, temperature, light, proximity, pressure, gravity, linear acceleration, rotation vector and gyroscope sensors*) on both ARDs and AVDs, we opted for a custom module, coined the Remote Mockup (RM) Library.

```
{
  'phone_id': 'SH0APPL00803'
  'user_id' : 'smartlab'
  [
   {
    't' : '1391198355678',
    's' : 'proximity',
    'v' : 0
     },
     {
     't' : '1391198355678',
     's' : 'accelerometer',
     'v' : [ '9.00','1.23','4.65']
     },
     {
     't' : '1391198355678',
     's' : 'rotation rate',
     'v' : [ '0.00','0.00','0.00']
     },
     {
     't' : '1391198355678',
     's' : 'orientation',
     'v' : [ '31.0','6.0','60.0']
     },
     {
     't' : '1391198355678',
     's' : 'rotation vector',
     'v' : 0
      },
 ]
}
```

```
 {
't' : '1391198355678',
's' : 'light',
'v' : 0
 },
 {
 't' : '1391198355678',
 's' : 'gyroscope',
 'v' : [ '0.0','0.0','0.0']
 },
 {
 't' : '1391198355680',
 's' : 'gravity',
 'v' : [ '0.00','0.00','0.00']
 },
 {
 't' : '1391198355678',
 's' : 'magnetic field',
 'v' : [ '-42.76','27.27','-8.58']
 },
 {
 't' : '1391198355678',
 's' : 'temperature',
 'v' : 0
  },
]
}
```

Fig. 8 Sensor/GPS Mockup (RM): (left, center) A data trace of various sensor measurements encoded in JSON. The given file can be loaded to ARDs and AVDs through this subsystem; **(right)** An application built with SLSensorManager using the measurements.

Our RM Library establishes a socket server on *DS* feeding devices with sensor or GPS readings encoded in JSON format and stores them in a big data repository. As this functionality is completely outside the ADB interaction stream, we were required to provide each application developer with a custom library, coined `SLSensorManager.jar`.

With Android Tools r18 and Android 4.0, developers have the opportunity to redirect real sensor measurements, produced by the ARDs, to the AVDs for further processing. It is important to mention that this functionality is the reverse of what we are offering. In our case, we want to be able to redirect data from a text file to an ARD, such that a given experiment on ARDs or AVDs uses a data file to drive its sensors. Recording sensor readings to text files can be carried out very easily with a variety of tools.

4 SmartLab Application

In this section, we provide some background information and definitions with respect to Radiomap-based indoor positioning systems in general, followed by a detailed description of our Anyplace indoor positioning system that has been utilized in our experiments to evaluate the different layers of SmartLab.

4.1 Background on Indoor Positioning

Radiomap-based indoor positioning systems carry out fine-grained localization with WiFi-based RadioMaps (i.e., 2-4 meters accuracy). To address the challenging signal propagation conditions that occur in indoor environments due to multi-path, reflections and diffractions, RSS fingerprints (i.e., vectors of RSS measurements

recorded from APs in the vicinity of the user) are collected a priori at predefined reference locations. Each RSS fingerprint is associated with the respective reference location and is stored in the so called *Radiomap* that covers the whole area of interest. Essentially, the Radiomap is a mapping from the multi-dimensional RSS space to the physical coordinates.

For example, assume an area A covered by a set of WiFi APs ($C = AP_1, AP_2, ..., AP_M$. A is not necessarily continuous and there is not a specific distribution of APs in the C set. Each Access Point AP_i of set C has one unique ID (i.e., BSSID/MAC address) that is publicly available and broadcasted. In the real world, A can be the joined areas of all indoor buildings at any requested resolution (e.g., campus, town, city, country or globe.) A smartphone user u inside A can use its smartphone's WiFi antenna to measure the *Received Signal Strength (RSS)* of any $AP_i^u \in C$ in its vicinity and to receive its ID. Each user u can also communicate with the server s over WiFi or 3G to forward its current RSS vector and receive its current location. For the localization to take place, s needs to have the *Radiomap* of A. A *Radiomap* is a matrix that expresses the relation between a location (x, y) and the RSS values of each $AP_i \in C$ measured at that particular (x, y). Let s maintain a 2-D matrix *MATRIX[N][M]*, which records the RSS value of M APs at N geo-locations *(x,y)*. For example, the radiomap *MATRIX* can be of the following format:

```
      Radiomap (MATRIX)
AP_1, AP_2, ... AP_M => x1,y1
AP_1, AP_2, ... AP_M => x2,y2
AP_1, AP_2, ... AP_M => x3,y3
...
AP_1, AP_2, ... AP_M => xN,yN
```

MATRIX is typically constructed by centrally overlaying several RSS vectors:

```
AP_1, AP_2, ... AP_l => xi,yi
```

with ($l << M$), which are recorded by users in motion (i.e., mobile crowd) that search for Wi-Fi wireless networks using a portable computer or smartphone. Additionally, *MATRIX* is extremely large with respect to N, as the M dimension is usually smaller and can be represented efficiently with adjacency-matrix structures. In our case, the areas are structured in a massively distributed NoSQL database. For ease of exposition, let *MATRIX* be denoted as a 2-D matrix where most points are null, e.g., *NaN* (i.e., a sparse matrix).

4.2 Anyplace Indoor Information System

Anyplace (see Figure 9) is a Radiomap-based indoor positioning and navigation platform that operates on top of Google Maps with a big data management Web 2.0 back-end service. Anyplace allows entities (i.e., users, companies, organizations, etc.) to realize indoor information management systems, including product search and point of interest (POI) navigation, on top of existing wireless network infrastructure by leveraging rich multi-sensory data available on smartphones. It consists of three major components: the *Server*, the *Architect website* and the

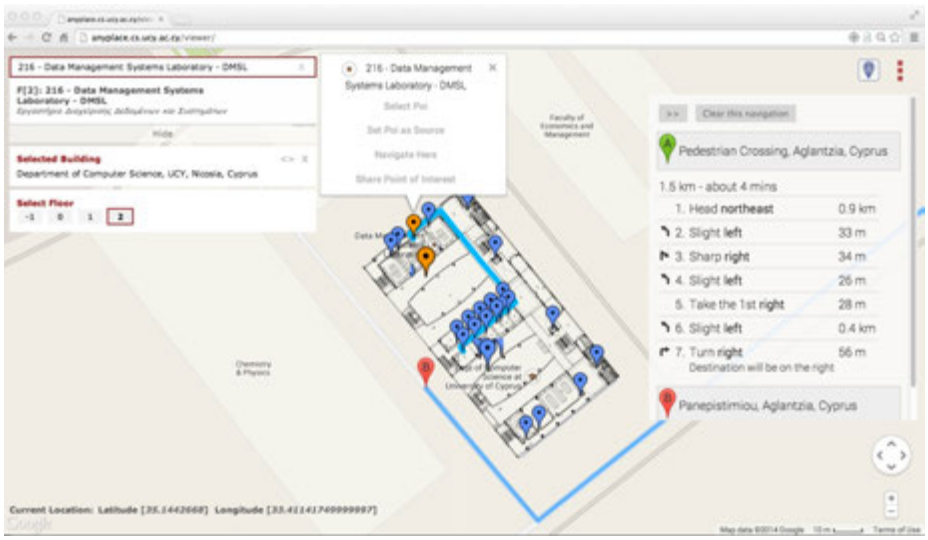


Fig. 9 AnyPlace: The user interface of the AnyPlace Viewer.

Client application on Android smartphones. The Anyplace Server follows a powerful big data architecture and provides a Web2.0 API that connects to the back-end database in order to efficiently store meta data regarding indoor POIs and provide navigation instructions to the end-user. The *Anyplace Architect* website offers a user-friendly interface for placing the blueprint of a building on top of Google Maps with multi-floor support. The user can later add, annotate and geo-tag POIs inside the building and also connect them to indicate feasible paths among POIs and then see the results through the *Anyplace Viewer*.

The *Anyplace Client* for Android smartphones has two modes and operates either as a *Logger* or as a *Navigator*. In *Logger mode*, the users may record signal strength information from nearby WiFi access points and contribute the collected data in a crowdsourcing fashion, by uploading them to the Anyplace Server through the API. These data are stored in the database and comprise the so-called Radiomap, which is later used for the provision of WiFi location information in GPS-deprived indoor environments. The *Navigator* is the main mode of operation that allows users to see their current location on top of the floorplan map and navigate between POIs inside the building. The onboard smartphone sensors (i.e., accelerometer, gyroscope and digital compass) are seamlessly integrated in our tracking module to smooth the WiFi locations and enhance the navigation experience.

Anyplace is chosen as the base application for our experimentations that follow since, as previously described in Section 4, it has the possibility to provide access to massive amounts of data (e.g., about 10 GBs of raw RSS fingerprint measurements per mapped building and sensory readings.) Additionally, debugging and testing an application similar to Anyplace is a time-consuming process and requires a lot of resources in terms of personnel and budget. Similarly, evaluating different algorithms and approaches / architectures requires a lot of time and the ability to repeatedly execute the same procedure while isolating and allowing fine-

grained control for all other variables (e.g., mobile device WiFi card, number of surrounding WiFi APs, etc).

5 Experimental Methodology

In this section, we describe our experimental methodology, which allows us to test many of the exciting features of SmartLab mentioned in the previous sections. We start-out by presenting the experimental testbed, the algorithms and datasets, and the hardware infrastructure utilized in the experiments.

5.1 Experimental Testbed

In this subsection, we describe the different scenarios and benchmark systems used in our experimental study to evaluate the performance of SmartLab testbed as a whole or particular SmartLab components.

Experimental Scenarios: Our experiments focus on several quantitative and qualitative dimensions: i) Experimental Series 1 - Deploying experiments with SmartLab, focuses on aspects of deploying and installing applications. Firstly, we evaluate the time efficiency of various smartphone connection modalities and then we compare SmartLab against two well-known and commercially available systems, namely the Remote TestLab and PerfectoMobile testbeds; ii) Experimental Series 2 - Experiments repeatability with SmartLab, focuses on re-programming and repeatability aspects of experiments on smartphones. We evaluate the time efficiency of three indoor positioning algorithms (i.e., KNN, MAP, MMSE) on SmartLab (with different number of devices) and compare it with the OpenIntents Sensor Simulator using the Anyplace indoor positioning engine; iii) Experimental Series 3 - Managing/Monitoring big data experiments, demonstrates how SmartLab collaborates with the big data and the algorithms repositories for the execution of various indoor positioning algorithms on a number of different devices. We also demonstrate how log traces from a completed experiment can be pushed back to the big data repository for further research; iv) Experimental Series 4 - Interacting with the Remote Mockup Library, demonstrates the advantages of utilizing the the Remote Mockup Library compared to real sensor devices using the Dead Reckoning (DR) indoor positioning algorithm and the Anyplace tracker; and v) Experimental Series 5 - Facilitating new research directions, demonstrates how SmartLab facilitates a new research direction through an experiment that utilizes crowd-sourced data to verify/filter data before storing them in SmartLab's big data repository.

Benchmark Systems: For assessing the performance of SmartLab we utilized Samsung's Remote Test Lab and PerfectoMobile testbed as well as the OpenIntents Sensor Simulator, using the same experimental settings. Samsung's Remote Test Lab and PerfectoMobile testbed are two of the most well-known commercially available smartphone testbeds. Both of them support functionalities similar to what SmartLab offers, thus it is advantageous to compare some of the key

elements, such as file upload and application installations, used during the experimental series 1. Similarly, the open source OpenIntents Sensor Simulator was selected in experimental series 2 because it supports similar functionality to SmartLab's Remote Mockup library. In summary, we use the following systems in our benchmarks:

Samsung's Remote Test Lab [17]: allows users to install and test applications over the web. It requires a Javascript enabled browser and a Java Runtime Environment. Users are able to reserve Samsung Android or Tizen devices over the web using a Java applet. Unfortunately, Samsung's Remote Test Lab does not offer interaction with multiple devices at once thus the repeatability of experiments is almost impossible. Furthermore, it does not allow any fine grained control over the reserved devices and it does not offer any sensor related functionalities.

PerfectoMobile Testbed [18]: allows users to install and test applications over the web on Android, iOS and Windows devices. Users are able to record and replay their sessions to heterogenous devices, but unfortunately it does not allow the interaction with multiple devices at once. On the other hand repeatability can be achieved by recording interactions with the reserved devices and using these recordings to perform the same interactions on other devices.

The OpenIntents Sensor Simulator: lets you simulate sensor data, similarly to SmartLab's Remote Mockup library, in an offline manner. Moreover, you can simulate your battery level and your GPS position using a telnet connection. It also provides features like editing, saving, loading and re-playing the scenario. It currently supports accelerometer, compass, orientation, temperature, light, proximity, pressure, gravity, linear acceleration, rotation vector and gyroscope sensors.

For ease of experimentation, we extended the functionality of OpenIntents Sensor Simulator to support the simulation of WiFi APs Receive Signal Strength (RSS). The major difference between the two applications / libraries is that the OpenIntents Sensor Simulator operates offline, records sensor data in XML and can only replay the pre-recorder scenario to only one device at a time as opposed to SmartLab's Remote Mockup library that stores recorded sensor data in JSON format online in a NoSQL database and replays the pre-recorded scenario to multiple devices at once. In addition, OpenIntents Sensor Simulators has to transfer the whole XML document created during the recording phase to the device prior simulation. On the other hand, SmartLab's Remote Mockup library uses a "sliding window" technique for transferring data between the devices and the server thus it does not require the whole JSON document to be transferred to the devices a priori.

5.2 Algorithms and Datasets

Algorithms: We have implemented several Radiomap-based indoor positioning algorithms, including the deterministic K-Nearest Neighbor (KNN) [37, 38], as well as the probabilistic Maximum A Posteriori (MAP) and Minimum Mean Square Error (MMSE) [39, 40]. These algorithms have been re-packaged in stand-alone,

plug-and-play Android libraries with related documentation and have been contributed to the SmartLab's Algorithm Repository. Similarly, the Dead Reckoning (DR) algorithm utilized in subsection 6.4 was also contributed in the Algorithm Repository in a similar fashion.

Datasets: In this work, we utilized the following datasets for our experiments.

UCY RadioMap: This radiomap is designed by data collected in a typical building at the Computer Science (CS) department of the University of Cyprus. We used a variety of Android smartphone devices (HTC Hero, HTC Desire, Samsung Nexus S, Motorola Xoom, HTC One X, etc) to collect 30 RSS fingerprints (i.e., RSS values of APs at a reference location) at 1500 distinct locations for a total of 45,000 reference fingerprints. There are 120 WLAN APs installed in the four (4) floors of this building including the APs of neighboring buildings that can be partially detected in different sections of the CS building. On average, 10.6 APs are detected per location. We collected our data by walking over a path that consists of 2900 locations. This dataset is utilized in Experimental Series 2, 3 and 5 of Section 6.

UCY Sensory Readings: The sensory readings contributed to the big data repository is designed by trajectories collected in the CS Department building at the University of Cyprus. Similarly, to the UCY RadioMap, we used variety of Android smartphone devices and over the years we have collected more than 20 km of trajectories of students walking in different section of the CS building. This dataset is utilized in Experimental Series 3 and 4 of Section 6.

5.3 Hardware Infrastructure

Back-end: Our evaluation is carried out on the DMSL VCenter[6] IaaS datacenter (i.e., private cloud), which encompasses 5 IBM System x3550 M3 and HP Proliant DL 360 G7 rackables featuring single socket (8 cores) or dual socket (16 cores) Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, respectively. These hosts have collectively 300GB of main memory, 16TB of RAID-5 storage on an IBM 3512 and are interconnected through a Gigabit network. The datacenter is managed through a VMWare vCenter Server 5.1 that connects to the respective VMWare ESXi 5.1 hosts. The computation cluster, deployed over our VCenter IaaS, comprises of 4 Ubuntu 12.04 (2 as SmartLab's Device Server and 2 for Couchbase NoSQL document store) server images, each featuring 8GB of RAM with 2 virtual CPUs (@ 2.40GHz). The images are equiped with fast local 10K RPM RAID-5 LSILogic SCSI disks, formatted with VMFS 5.54 (1MB block size).

Big Data Repository: Our repository, outlined in subsection 3.6, is configured in a demilitarized network zone that features a HAProxy[7] HTTP load balancer to distribute the load to respective SmartLab's Device Servers. Each SmartLab's Device Server features an Apache HTTP server and can access the two Couchbase NoSQL document store[8] servers for storing the sensor data collected by our users.

[6] DMSL VCenter @ UCY. `http://goo.gl/dZfTE5`

[7] HAProxy. `http://haproxy.1wt.eu/`

[8] Couchbase. `http://www.couchbase.com/`

As mentioned before, Couchbase, stores data across the cloud in JSON format, which can be indexed, queried and directly exposed to the SmartLab's smartphones devices through our custom API hosted in each SmartLab Device Server.

Smartphone Devices: Table 1 presents a comprehensive list and a categorization, according the number of cores, of all devices used during the experimentation phase. A variety of other diverse models is available for reservation and experimentation through SmartLab.

Table 1 Device Specification used for the experiments

Category	**Target Device Model**	**CPU**	**Android OS version**
Single-core (1GHz)	HTC Desire	Qualcomm QSD8250 Snapdragon	Jelly Bean (v 4.2.2)
Dual-core (1GHz)	Motorola Xoom	Nvidia Tegra 2 T20	Ice Cream (v.4.0.4)
Quad-core (1.5GHz)	HTC One X	Nvidia Tegra 3	Jelly Bean (v4.2.2)
	Google Nexus 7	Qualcomm Snapdragon S4Pro	Kit Kat (v4.4.2)

6 Experiments

In this section, we present an experimental study that demonstrates how SmartLab facilitates big data experiments on smartphones in an efficient and effective manner.

6.1 **Experimental Series 1:** Deploying Experiments with SmartLab

In the first experimental series, we focus on aspects of the deployment phase. More specifically, we measure the time of various connection modalities and the time required for an algorithm (deployed in an application) and its related files to be *deployed* and *installed* on a number of target devices.

Connectivity Experiment: In order to evaluate the time-efficiency of various connection modalities (i.e., wired or wireless) to our DS, we have performed an experiment using wired ARDs (i.e., ARD-Local and ARD-Remote) and wireless ARDs (i.e., ARD-WiFi). The wireless connectivity is handled by a 802.11b/g/n wireless router (max. 300 Mbps) deployed in the same room as the ARDs and connected directly to the *DS*.

These experiments were conducted for calculating the time needed for transferring 2.5MBs (i.e., the size of a medium Android application and the applications used in the experimentation phase) to up to 16 devices. In our experimentation, we observed that ARD-WiFi features the worst time compared to the other two alternatives. For example, in the case of 16 ARDs, the time required for sending the file reaches 12 seconds as opposed to 4.8 seconds and 1.4 seconds for ARD-Remote

and ARD-Local, respectively, as this is summarized in Table 1. One reason for this is because the cascading USB 2.0 hubs offer much higher transfer rate (max. 480Mbps) than the wireless router, which never reached over 130Mbps.

Table 2 Transferring a 2.5MB file to 16 Devices

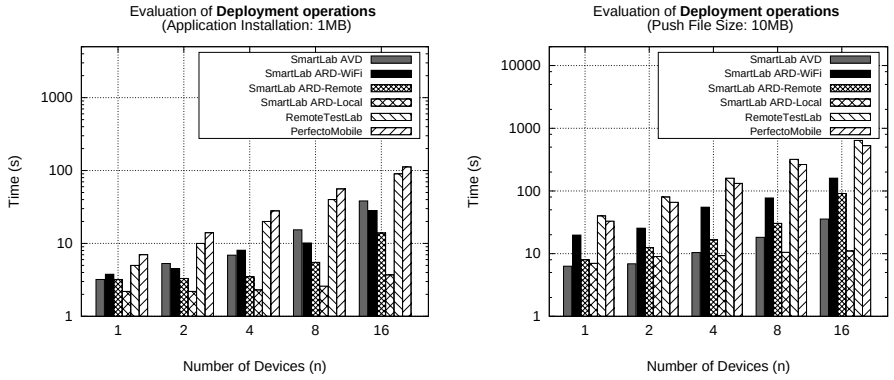
Connectivity Mode	Average Time (10 trials)
ARD-Local	1.4 seconds
ARD-Remote	4.8 seconds
ARD-WiFi	12 seconds

Another observation is that ARD-Local devices outperform ARD-Remote devices, as the former are locally mounted to *DS*, thus avoid the overhead of transferring data via a network.

Deployment Experiment: The time required to deploy files/applications from and to target devices differs significantly according to the type of device. In order to investigate this, we have conducted an experiment that measures the time overhead for transferring files to/from the aforementioned different types of target devices. More specifically, we have utilized a 10MB file and distributed this file to up to 16 AVDs, 16 ARD-WiFi, 16 ARD-Remote and 16 ARD-Local, individually. The ARD-WiFi devices were assigned to students that were moving around our department premises in order to provide a realistic mobility scenario. Each experiment was executed 10 times and we recorded the average at each attempt. Moreover, we are benchmarking the performance of our proposed SmartLab testbed against two commercially available testbeds, namely Samsung's Remote Test Lab and PerfectoMobile, by measuring the required amount of time for (i) transferring a 10 MB file to multiple remotely managed Android devices and (ii) installing a 1MB application.

Firstly, the results on the left side of Figure 10 clearly illustrate the advantage of using ARD-Local devices in experiments requiring large amounts of data to be transferred to devices (e.g., large trajectory datasets). Additionally, the results show that the disk I/O overhead introduced by the usage of the emulated devices (i.e., AVDs) justifies the linearly increasing amount of time for transferring files on those devices. In the case of remotely connected ARDs (ARD-Remote) the large time delays are attributed to communicating over the network. Finally, the ARD-WiFi devices feature the worst time overhead because the file transfer is hampered by the wireless network's low bandwidth in mobility scenarios.

Installation Experiment: In order to examine the cost of installing applications, which includes transferring the application file (.apk) and its installation, we have conducted another experiment that calculates the required time. Similarly to the previous experimental setting, we measure the time for transferring and installing a sample application of typical 1MB size, to each type of target devices. The results are shown on the right side of Figure 10. We observe that transferring and installing the selected sample application introduces an additional time overhead. For example, in the 1x target device scenario, the sample application requires a total of ≈2.2s from which 0.7s accounts for file transfer and 1.5s for installing the application. The results provide a clear indication that emulated de-



(a) Deployment

(b) Installation

Fig. 10 Deployment Experiment. Evaluating the average time for deploying files and installing applications on different types of target devices on SmartLab as well as on Samsung's Remote Test Lab and PerfectoMobile testbed.

vices are not the appropriate type of Android devices for performing I/O intensive experiments such as evaluating network performance and database benchmarking. Additionally, the sample application utilized in the experiments did not perform any specialized deployment functions during setup (e.g., extracting other files, installing database), thus its installation overhead is minimal. The time required for installing more complex applications varies greatly according to the requirements of the application.

Finally, Figure 10 clearly demonstrates the superiority of SmartLab with respect to Samsung's Remote Test Lab and PerfectoMobile testbed in terms of the required time for transferring files and remotely installing applications to multiple devices. The reason behind this is the functionality provided by SmartLab to upload single files to multiple devices at once. Similarly, SmartLab provides the ability of installing applications to multiple devices at once while the aforementioned commercial testbeds allow the user to manually install applications only to a single device at a time. Additionally, we were limited to a 10 MB file upload for the experimentation series since the PerfectoMobile testbed limits the size of the upload file to 10 MBs while SmartLab allows files upto the size of the user's home directory.

6.2 **Experimental Series 2:** Experiments Re-programming/Repeatability

In order to examine how SmartLab can facilitate re-programming and repeatability of experiments on a large number of target devices (heterogeneous or not), we have conducted an experiment that calculates the required time for executing a variety of indoor positioning algorithms on single or multiple devices through

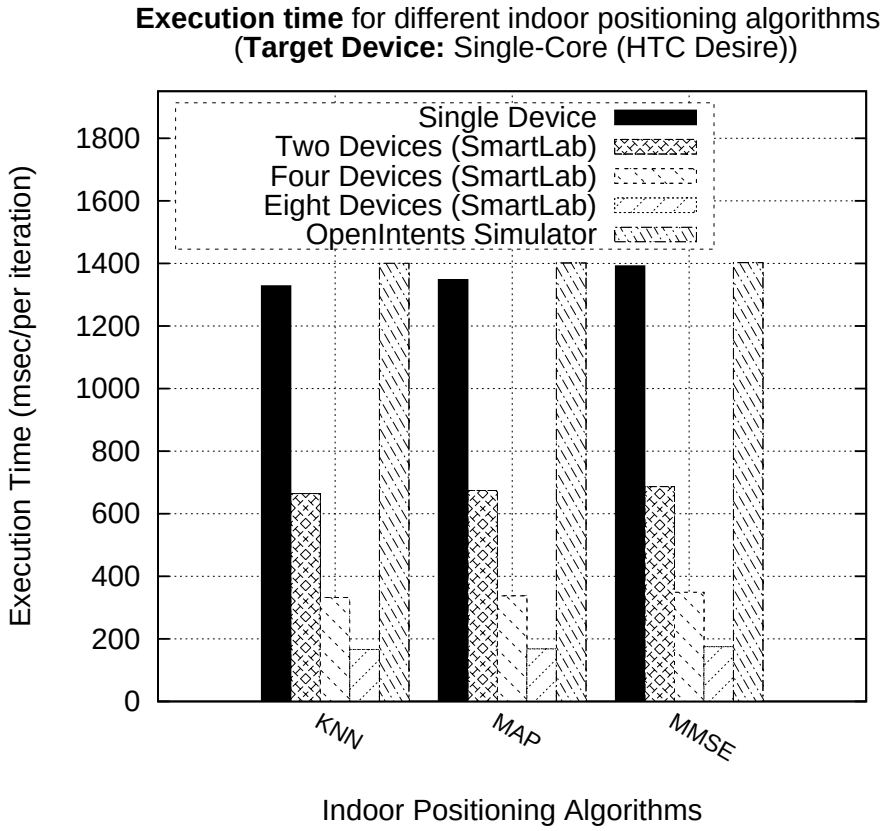


Fig. 11 Re-programming / Repeatability Experiment. Evaluating the average time for retrieving 450 times the position of the user on different types of target devices using different indoor positioning algorithms on SmartLab and the OpenIntents Sensor simulator.

SmartLab and compare it with the OpenIntents Sensor Simulator. Here it is important to notice that we updated the OpenIntents Sensor Simulator and added the capability of recording WiFi Receive Signal Strength (RSS) for comparison purposes. Similarly to the previous experimental settings, we measure the time, in seconds, required for executing an indoor positioning algorithm (i.e., retrieving the geo-location of the user) on a varying number of target devices. Each experiment was executed 5 times consisting of bunches of 450 iterations on a single-core target device (i.e., HTC Desire) and we recorded the average of each bunch.

The results are illustrated in Figure 11, which clearly illustrates the advantage of utilizing SmartLab in the context of these type of experiments since it allows the massive parallelization of the indoor positioning algorithm execution on multiple target devices, resulting in a significant reduction of the amount of time required for experimentation. Additionally, another observation was the ability to feed the exact WiFi fingerprints from the big data repository to the target devices in order to guarantee the identical execution of the experiments and reduce any possible side-effects.

Finally, another important observation was that the simulators require different amount of time, as opposed of what might be expected. The difference in the time needed for executing an algorithm is due to the fact that OpenIntents Sensor Simulator requests the whole amount of data before the beginning of an experiment repetition (and thus it requires more time), while SmartLab requests only a small chunk at the beginning of the experiment and then requests more data while the repetition is in progress.

6.3 **Experimental Series 3:** Managing/Monitoring Big Data Experiments

In this experimental series, we focus on the qualitative features of SmartLab and show how these enable researchers to manage and monitor their experiments more efficiently. We first show an experiment that demonstrates how SmartLab can collaborate with the big data and the algorithms repositories for the execution of various indoor positioning algorithms on a number of different devices. Secondly, we demonstrate how log traces from a completed experiment can be pushed back to the big data repository for further investigation.

Monitoring big data Experiments: In the first experiment, we utilize a dataset of WiFi fingerprints collected over the years from the crowd in the context of the Anyplace project. This data is stored in our big data repository in a structured manner so that querying of results can be facilitated in an easy manner for ongoing research activities. Additionally, we deploy the KNN, MAP and MMSE algorithms in the AnyPlace indoor positioning engine and configure SmartLab for parallel monitoring of CPU utilization, Power consumption, RAM usage and Disk I/O. Moreover, we port all experiments on a number of heterogeneous smartphone devices simultaneously. More specifically, we utilize smartphone devices with different CPU cores in order to investigate how the clock speed of the CPU core might affect the execution time. Each experiment was executed 10 times on a single-core target device (i.e., HTC Desire), a dual-core target device (i.e., Motorola XOOM) and two quad-core devices (i.e., Nexus 7 and HTC One X). The results for the average execution time for each iteration are shown in Figure 12. We omit the results for the other parameters as similar observations apply.

The results clearly illustrate that the available number of cores affect the execution time of each algorithm. For instance, the dual-core target device was 16.7% faster than the single core device, the first-quad core device (i.e., Nexus 7) was 15% faster than the dual-core device and the second quad-core device (i.e., HTC One X) was about 35% faster than the dual-core device. Surprisingly, even though both quad-core target devices run at the same clock speed (i.e., 1.5 GHz), they have a different chipset (i.e., HTC One X - Nvidia Tegra 3 and Nexus 7 - Qualcomm Snapdragon S4Pro) the results shown that the first one is significantly faster than the second one (about 20%). Consequently, these findings illustrate how the CPU chipset negatively or positively affects the execution time.

Fine-grained Logging and Repository Updating: We also show how SmartLab provides fine-grained logging by measuring the CPU utilization and power consumption required during positioning on heterogeneous devices. Similar to

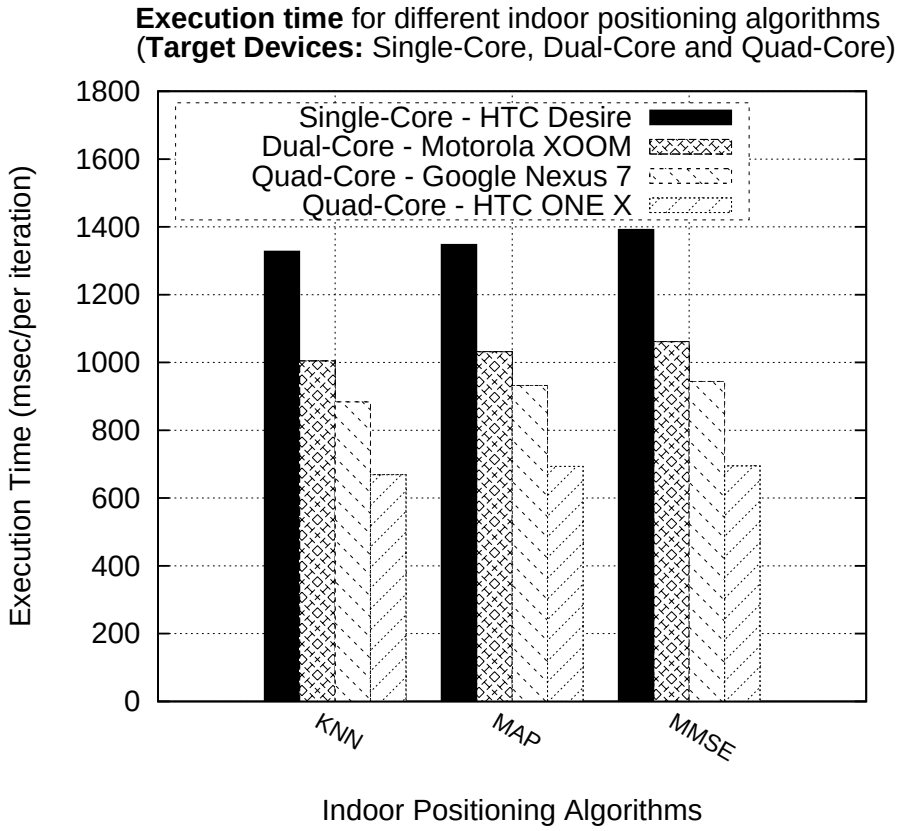
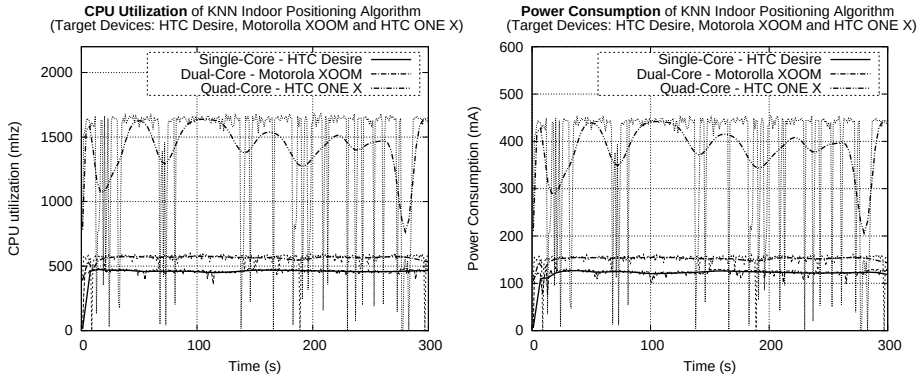


Fig. 12 Execution Time. Evaluating the average time for executing a variety of indoor positioning algorithms on different types of target devices.

the previous setup, we repeat each experiment 10 times and we show how the cpu/power logs are analyzed and visualized through SmartLab.

The results are presented in Figure 13. For presentation reasons, we only show the results for the KNN algorithm on each type of device as similar results apply for the other settings. Additionally, please keep in mind, that a certain amount of time is presented (i.e., 300 seconds), thus it is not correct to derive the conclusion that the quad-core device (i.e., HTC One X) requires more CPU and power resources in comparison to the other target devices since someone has to consider the fact that in the same amount of time the quad-core device will provide almost two times (2x) the amount of algorithm execution iterations in comparison to a single-core device and about 35 % more in comparison to a dual-core device.

The above findings illustrate the significance of mobile testbeds, similar to SmartLab, which provide fine-grained control and logging over heterogeneous devices and could facilitate further research and experiments. Finally, SmartLab contributes the results (i.e., CPU utilization traces, power consumption traces) from the above experiments, back to big data repository since they could facilitate further research in related areas.



(a) F1 (b) F2

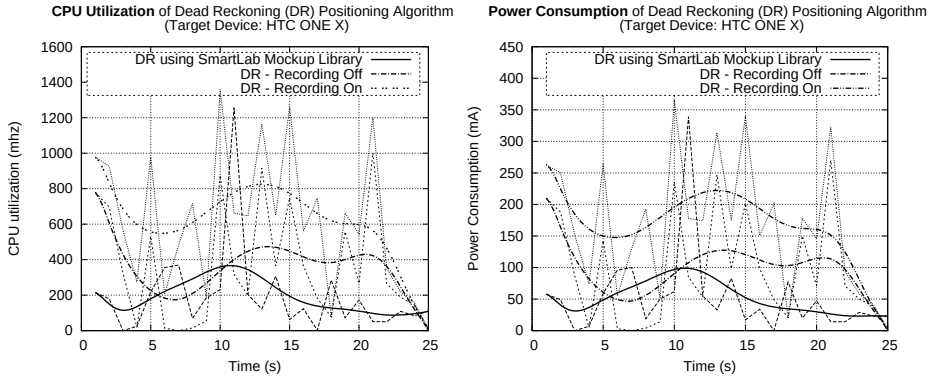
Fig. 13 SmartLab Fine-Grained Logging. Logging the CPU utilization (Sub-Figure F1) and Power consumption (Sub-figure F2) for the KNN Algorithm on different types of target devices

6.4 **Experimental Series 4:** Interacting with the Remote Mockup Library

In this experimental series, we demonstrate utilization of the in-house developed Remote Mockup Library by seamlessly integrating GPS readings in the Anyplace Tracker application. The main objective is to illustrate how the Remote Mockup Library feeds the Anyplace Tracker with previously collected sensor readings in order to evaluate the efficiency of the Dead Reckoning (DR) algorithm used for indoor positioning using sensory readings.

Similarly to the experimental setting of section 6.2, we examine the CPU utilization and power consumption of the DR algorithm on a quad-core device (i.e., HTC One X). Each experiment was executed 10 times using three different modes: i) DR algorithm in collaboration with SmartLab's Remote Mockup Library, ii) DR algorithm as available in the Anyplace Tracker (DR-Recording Off); and iii) DR algorithm while recording and contributing the sensory readings to the big data repository (DR-Recording On). Finally, we analyzed the resulted PowerTutor log files through SmartLab.

The results are presented in Figure 14 and illustrate the CPU utilization and power consumption of each mode. The results indicate that the usage of sensors instead of retrieving sensory data from a remote big data repository is more CPU and power intensive ($\approx 55\%$). Finally, similarly to the previous experiment, all results (i.e., CPU utilization traces and Power consumption traces) were contributed to the big data repository.



(a) F1 (b) F2

Fig. 14 SmartLab Fine-Grained Logging. Logging the CPU utilization (sub-figure F1) and Power consumption (sub-figure F2) for the Dead Reckoning Algorithm using different modes.

6.5 **Experimental Series 5:** Facilitating New Research Directions

In this experimental series, we demonstrate a new research direction that attempts to verify crowd-sourced indoor positioning data before storing them into the big data repository. To accomplish this, we have conducted an experiment consisting of two phases: i) The calculation of the *Average Positioning Error (APE)* and its standard deviation for each available indoor positioning algorithm. ii) Use of the previously calculated APE in order to filter possible malicious contributions (e.g., intentionally or accidentally malicious contributions).

Phase 1 - Calculating Average Positioning Error: In order to calculate the APE for each available indoor localization algorithm, we evaluated the error resulted by the calculation of the Euclidean distance on the position returned by the algorithm and the pre-verified position of the user (ground truth). For instance a vector of WiFi MAC addresses and RSS fingerprints were provided as input in each algorithm from a pre-verified known location (e.g., x1, y1) and the APE error was the Euclidean distance of the verified position and the resulted by the algorithm position (e.g., x2, y2).

The results are presented in Table 4 and illustrate the APE and the standard deviation calculated for each available positioning algorithm.

Table 3 Calculated Average Position Error for Available Indoor Positioning Algorithms

Algorithms	Average Positioning Error	Standard Deviation
KNN Algorithm	10.12 meters	5.50 meters
MAP Algorithm	25.73 meters	25.57 meters
MMSE Algorithm	9.94 meters	9.71 meters

Phase 2 - Filtering malicious contributions: In order to examine the efficiency of the above proposed technique we decided to "feed" every contributed vector of the user's geo-position, MAC addresses and RSS fingerprints to SmartLab. Subsequently, SmartLab was responsible to provide the aforementioned vector as input to a number of heterogeneous devices available for running the indoor positioning algorithms. Finally, SmartLab was responsible for collecting the results and by utilizing a majority vote algorithm, it concluded if the contributed data was malicious or not before contributing them to the big data repository. In order to prepare the experiment, we collected a sample of 90 fingerprints and we manually classified them in two categories. Half of the fingerprints were labeled as our "verified" test dataset and contained only valid information, while the other half fingerprints were manipulated and labeled as the "malicious" contributions. As a final step, we streamed the two datasets to SmartLab and observed the validation process.

Table 4 Calculated Average Position Error for Available Indoor Positioning Algorithms

Dataset	# of Fingerprints Contributed	Correctness
Verified dataset	41 out of 45	91 %
Malicious dataset	6 out of 45	86 %

The results illustrated in Table 4, indicate that SmartLab, can play a key role for crowd-sourced data verification. More specifically, the results show that only 41 out of the 45 fingerprints from the "verified" dataset (about 91%) were correctly contributed to the big data repository while only 4 out of 45 (9%) were mistakenly rejected. Similar observations are derived from the results for the "malicious" dataset since only 6 out of 45 (14%) of the fingerprints were mistakenly contributed to the big data repository while 39 out of 45 (86%) of the fingerprints were correctly rejected.

Further research should be conducted to investigate and reduce the amount of false positives. We assume that with the contribution of more open source positioning algorithms to the Algorithms repository by other research groups, we will be able to decrease the false positives and increase the accuracy of the propose technique.

7 Conclusions and Future Work

In this paper, we have presented the SmartLab testbed and demonstrated how it can transparently manage the complexity of large-scale data management experiments on real smartphones. We have presented and validated the qualitative features of the SmartLab architecture through experiments on a complex Indoor Positioning System for smartphones that we have developed in-house and have shown how researchers can perform their experiments in a more effective and efficient manner. We have also shown that the proposed SmartLab testbed performs better than the well-known and commercially available Samsung's Remote Test Lab, PerfectoMobile testbed and the OpenIntents Sensor Simulator with respect to several quantitative and qualitative metrics.

In the future, we plan to facilitate testing of algorithms, protocols and application in mobile urban environments, providing an open mobile programming cloud to the community. Additionally, we plan to enhance SmartLab with support for federated experiments by allowing groups around the globe to interface with SmartLab, enabling a truly global smartphone programming cloud infrastructure. Moreover, SmartLab can be utilized in the context of projects using replicated execution [41] for validation and verification purposes. Finally, it would be interesting to extend SmartLab to perform more dynamic deployment and re-configuration similarly to (or through) the OSGI and/or Kalimucho platforms.

Acknowledgments: We would like to thank the anonymous reviewers for their valuable comments and suggestions. This work was supported in part by the last author's Startup Grant, funded by the University of Cyprus, COST Action IC903 (MOVE) "Knowledge Discovery for Moving Objects" EU's FP7 (MODAP) "Mobility, Data Mining, and Privacy" projects, as well as an industrial grant by MTN Cyprus.

References

1. Georgios Chatzimiloudis, Andreas Konstantinidis, Christos Laoudias and Demetris Zeinalipour-Yazti, "Crowdsourcing with Smartphones." Internet Computing, IEEE, vol.16, no.5, pp.36,44, Sept.-Oct. 2012.
2. Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh and Reynold Xin, "CrowdDB: answering queries with crowdsourcing." In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (*SIGMOD '11*). ACM, New York, NY, USA, 61-72., 2011.
3. Adam Marcus, Eugene Wu, Samuel Madden and Robert C. Miller, "Crowdsourced Databases: Query Processing with People," In Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (*CIDR '11*) January 9-12, 2011.
4. David R. Choffnes, Fabian Bustamante and Zihui Ge, "Crowdsourcing service-level network event monitoring," In Proceedings of the ACM SIGCOMM 2010 conference (*SIGCOMM '10*). ACM, New York, NY, USA, 387-398, 2011.
5. Andreas Konstantinidis, Demetrios Zeinalipour-Yazti, Panayiotis G. Andreou, Panos K. Chrysanthis, and George Samaras. *"Intelligent Search in Social Communities of Smartphone Users"*, Distributed and Parallel Databases, Springer US, Volume 31, 115-149, 2013.
6. Archak Money, "Glory and Cheap Talk: Analyzing Strategic Behavior of Contestants in Simultaneous Crowdsourcing Contests on TopCoder.com" In Proceedings of the 19th international conference on World Wide Web (*WWW '10*). ACM, New York, NY, USA, 21-30, 2011.
7. Omar F. Zaidan and Chris Callison-Burch, "Crowdsourcing Translation: Professional Quality from Non-Professionals." In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (*ACL '11*), Stroudsburg, PA, USA, 1220-1229, 2011.
8. Anthony Brew, Derek Greene and Padraig Cunningham, "Using Crowdsourcing and Active Learning to Track Sentiment in Online Media," In Proceedings of the 19th European Conference on Artificial Intelligence (*ECAI '10*), Helder Coelho, Rudi Studer, and Michael Wooldridge (Eds.). IOS Press, Amsterdam, The Netherlands, The Netherlands, 145-150, 2010.
9. Georgios Larkou, Constantinos Costa, Panayiotis G. Andreou, Andreas Konstantinidis and Demetrios Zeinalipour-Yazti. *"Managing Smartphone Testbeds with Smartlab"*, In Proceedings of the 27th International Conference on Large Installation System Administration, USENIX Association, LISA'13, 115–132, 2013.
10. Christos Laoudias, George Constantinou, Marios Constantinides, Silouanos Nicolaou, Demetrios Zeinalipour-Yazti, and Christos G. Panayiotou. *"The Airplace Indoor Positioning Platform for Android Smartphones"*, In Proceedings of the 13th IEEE International Conference on Mobile Data Management (*MDM'12*), IEEE Computer Society, Washington, DC, USA, 312-315, 2012.

11. Georgios Larkou, Marios Mintzis, Stefano Taranto, Andreas Konstantinidis, Panayiotis G. Andreou, and Demetrios Zeinalipour-Yazti. "Sensor Mockup Experiments with SmartLab." In Proceedings of the 13th International Symposium on Information processing in sensor networks (*IPSN '14*). IEEE Press, Piscataway, NJ, USA, 339-340, 2014.
12. Stavros Harizopoulos, and Spiros Papadimitriou. *"A case for micro-cellstores: energy-efficient data management on recycled smartphones"*, In Proceedings of the Seventh International Workshop on Data Management on New Hardware (*DaMoN'11*), ACM, New York, NY, USA, 50-55, 2011.
13. Lambros Petrou, George Larkou, Christos Laoudias, Demetrios Zeinalipour-Yazti, and Christos G. Panayiotou. "Crowdsourced Indoor Localization and Navigation with Anyplace." In Proceedings of the 13th International Symposium on Information processing in sensor networks (IPSN '14). IEEE Press, Piscataway, NJ, USA, 331-332, 2014.
14. Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. *"A Blueprint for Introducing Disruptive Technology into the Internet"*, In Proceedings of the ACM SIGCOMM 2003 conference (*SIGCOMM '03*) 33, 1 (January 2003), 59-64, 2003.
15. David Johnson, Tim Stack, Russ Fish, Daniel Montrallo Flickinger, Leigh Stoller, Robert Ricci, and Jay Lepreau. *"Mobile Emulab: A Robotic Wireless and Sensor Network Testbed"*, In Proceedings of the 25th IEEE International Conference on Computer Communications (*INFOCOM'06*), IEEE Computer Society, Washington, DC, USA, 1-12, 2006.
16. Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. *"MoteLab: a wireless sensor network testbed"*, In Proceedings of the 4th international symposium on Information processing in sensor networks (*IPSN'05*). IEEE Press, Piscataway, NJ, USA, Article 68, 2005.
17. Samsung, Remote Test Lab: `http://goo.gl/p7SNU`
18. Perfecto Mobile, `http://goo.gl/DSlP9`.
19. Keynote Systems Inc., Device Anywhere: `http://goo.gl/mCxFt`
20. AT&T Application Resource Optimizer (ARO), Free Diagnostic Tool: `http://goo.gl/FZnXS`
21. Tim Verry. *"MegaDroid simulates network of 300,000 Android smartphones"*, Extremetech.com, Oct 3, 2012. `http://goo.gl/jMaS8`.
22. 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*). ACM, New York, NY, USA, 63-76, 2010.
23. Eduardo Cuervo, Peter Gilbert, Bi Wu, and Landon Cox. *"CrowdLab: An Architecture for Volunteer Mobile Testbeds"*, In Proceedings of the 3rd International Conference on Communication Systems and Networks (*COMSNETS'11*), IEEE Computer Society, Washington, DC, USA, 1-10, 2011.
24. Rishi Baldawa, Micheal Benedict, M. Fatih Bulut, Geoffrey Challen, Murat Demirbas, Jay Inamdar, Taeyeon Ki, Steven Y. Ko, Tevfik Kosar, Lokesh Mandvekar, Anandatirtha Sathyaraja, Chunming Qiao, and Sean Zawicki. *"PhoneLab: A large-scale participatory smartphone testbed (poster and demo)"*, In Proceedings of the 9th USENIX conference on Networked systems design & implementation (*NSDI'12*). USENIX Association, Berkeley, CA, USA, 2012.
25. Adam J. Oliner, Anand P. Iyer, Eemil Lagerspetz, Ion Stoica and Sasu Tarkoma. *"Carat: Collaborative Energy Bug Detection (poster and demo)"*, In Proceedings of the 9th USENIX conference on Networked systems design & implementation (*NSDI'12*). USENIX Association, Berkeley, CA, USA, 2012.
26. Jeffrey A. Hoffer, V. Ramesh, Heikki Topi, *"Modern Database Management*, 2013.
27. Smart Metering Entity website, http://www.smi-ieso.ca/mdmr, Jan, 2014.
28. Popular Science: Inside Google's Quest To Popularize Self-Driving Cars article. http://www.popsci.com/cars/article/2013-09/google-self-driving-car, Jan, 2014.
29. Chi Zhang, Feifei Li, and Jeffrey Jestes, *"Efficient parallel knn joins for large data in mapreduce"*, In Proceedings of the 15th International Conference on Extending Database Technology (*EDBT '12*), New York, NY, USA: ACM, 2012, pp. 38–49.
30. Wei Lu, Yanyan Shen, Su Chen, and Beng C. Ooi, *"Efficient processing of k nearest neighbor joins using mapreduce"*, In Proceedings of VLDB Endow., vol. 5, no. 10, pp. 1016–1027, Jun. 2012.
31. Ioannis Kitsos, Kostas Magoutis, and Yannis Tzitzikas. *"Scalable entity-based summarization of web search results using MapReduce"*, Distributed and Parallel Databases, Springer US, Volume 32, 405-446, 2014.

32. Hadoop website, http://hadoop.apache.org/, Jan, 2014.
33. Yanying Gu, Anthony Lo and Ignas Niemegeers, *"A survey of indoor positioning systems for wireless personal networks*, IEEE Communications Surveys & Tutorials, Vol.11, pp.13-32, 2009.
34. Chin-Lung Li, Christos Laoudias, Georgios Larkou, Georgios Chatzimilioudis, Demetrios Zeinalipour Yazti, and Christos G. Panayiotou, *"Hybrid Indoor Positioning on Multi-Sensor Android Smartphones*, In Proceedings of International Conference on Indoor Positioning and Indoor Navigation (*IPIN*) , Sydney, Australia, 2012.
35. Andreas Konstantinidis, Constantinos Costa, Georgios Larkou, and Demetrios Zeinalipour-Yazti. *"Demo: a programming cloud of smartphones"*, In Proceedings of the 10th international conference on Mobile systems, applications, and services (*MobiSys'12*). ACM, New York, NY, USA, 465-466, 2012.
36. Hyojun Kim, Nitin Agrawal, and Cristian Ungureanu. *"Revisiting storage for smartphones"*, In Proceedings of the 10th USENIX conference on File and Storage Technologies (*FAST'12*). USENIX Association, Berkeley, CA, USA, 17-31, 2012.
37. Paramvir Bahl and Venkata Padmanabhan, *"RADAR: an in-building RF-based user location and tracking system"*, in 19th IEEE Conference on Computer Communications, (*Infocom'00*), 2000, 775-784.
38. Binghao Li, James Salter, Andrew G. Dempster, and Chris Rizos, *"Indoor positioning techniques based on wireless LAN"*, In Proceedings of the 1st IEEE International Conference on Wireless Broadband and Ultra Wideband Communications, 2006.
39. Moustafa Youssef and Ashok Agrawala, *"The Horus WLAN location determination system,"*, In Proceedings of the 3rd international conference on Mobile systems, applications, and services (*MobiSys'05*). ACM, Seattle, WA, USA, 205-218, 2005.
40. Teemu Roos, Petri Myllymaki, Henry Tirri, Pauli Misikangas, and Juha Sievanen, *"A probabilistic approach to WLAN user location estimation"*, International Journal of Wireless Information Networks, vol. 9, no. 3, pp. 155-164, Jul. 2002.
41. Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. *"Paranoid Android: versatile protection for smartphones"*, In Proceedings of the 26th Annual Computer Security Applications Conference (*ACSAC'10*). ACM, New York, NY, USA, 347-356, 2010.