Managing Smartphone Testbeds with SmartLab

Georgios Larkou, Constantinos Costa, Panayiotis G. Andreou, Andreas Konstantinidis, and Demetrios Zeinalipour-Yazti, *University of Cyprus*

https://www.usenix.org/conference/lisa13/technical-sessions/papers/larkou

This paper is included in the Proceedings of the
27th Large Installation System Administration Conference (LISA '13).

November 3–8, 2013 • Washington, D.C., USA

ISBN 978-1-931971-05-8

Open access to the
Proceedings of the 27th Large Installation
System Administration Conference (LISA '13)
is sponsored by USENIX.

Managing Smartphone Testbeds with SmartLab

Georgios Larkou
Dept. of Computer Science
University of Cyprus
glarkou@cs.ucy.ac.cy

Constantinos Costa
Dept. of Computer Science
University of Cyprus
costa.c@cs.ucy.ac.cy

Panayiotis G. Andreou
Dept. of Computer Science
University of Cyprus
panic@cs.ucy.ac.cy

Andreas Konstantinidis
Dept. of Computer Science
University of Cyprus
akonstan@cs.ucy.ac.cy

Demetrios Zeinalipour-Yazti
Dept. of Computer Science
University of Cyprus
dzeina@cs.ucy.ac.cy

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. In this paper, we make three major contributions: First, we propose a comprehensive architecture, coined SmartLab[1], for managing a cluster of both real and virtual smartphones that are either wired to a private cloud or connected over a wireless link. Second, we propose and describe a number of Android management optimizations (e.g., command pipelining, screen-capturing, file management), which can be useful to the community for building similar functionality into their systems. Third, we conduct extensive experiments and microbenchmarks to support our design choices providing qualitative evidence on the expected performance of each module comprising our architecture. This paper also overviews experiences of using SmartLab in a research-oriented setting and also ongoing and future development efforts.

1 Introduction

Last year marked the beginning of the post PC era[2], as the number of smartphones exceeded for the first time in history the number of all types of Personal Computers (PCs) combined (i.e., Notebooks, Tablets, Netbooks and Desktops). According to IDC[3], Android is projected to dominate the future of the smartphone industry with a share exceeding 53% of all devices shipped in 2016. Currently, an Android smartphone provides access to more than 650,000 applications, which bring unprecedented possibilities, knowledge and power to users.

[1]Available at: `http://smartlab.cs.ucy.ac.cy/`
[2]Feb. 3, 2012: Canalys, `http://goo.gl/T81iE`
[3]Jul. 6, 2012: IDC Corp., `http://goo.gl/CtDAC`

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. To this end, we have implemented and demonstrated *SmartLab* [21], 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. Our current architecture is ideal for scenarios that require *fine-grained* and *low-level* control over real smartphones, e.g., OS, Networking, DB and storage [20], security [5], peer-to-peer protocols [22], but also for scenarios that require the engagement of physical sensors and geo-location scenarios [38],[24]. Our preliminary release has been utilized extensively in-house for our research and teaching activities, as those will be overviewed in Section 7.

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)*, as illustrated in Figure 1. 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. In this work, we present the anatomy of the SmartLab Architecture, justifying our design choices via a rigorous microbenchmarking process. Our findings have helped us enormously in improving the performance and robustness of our testbed leading to a new release in the coming months.

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 granular-

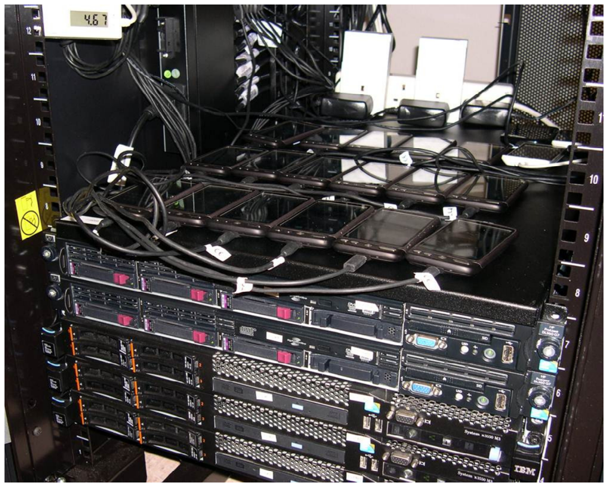


Figure 1: **Subset of the SmartLab smartphone fleet connected locally to our datacenter. More devices are connected over the wireless and wired network.**

ity (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 *Microcellstores* [16], 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. The contributions of this work are summarized as follows:

- **i) Architecture:** We present the architecture behind SmartLab, a first-of-a-kind open smartphone programming cloud that enables fine-grained control over both ARDs and AVDs via an intuitive web-based interface;
- **ii) Microbenchmarks:** We carry out an extensive array of microbenchmarks in order to justify our implementation choices. Our conclusions can be instrumental in building more robust Android smartphone management software in the future;
- **iii) Experiences:** We present our research experiences from using SmartLab in four different scenarios including: trajectory benchmarking [38], peer-to-peer benchmarking [22], indoor localization testing [24] and database benchmarking; and
- **iv) Challenges:** We overview ongoing and future developments ranging from Web 2.0 extensions to urban-scale deployment and security studies.

The rest of the paper is organized as follows: Section 2 looks at the related work, Section 3 presents our SmartLab architecture, while subsequent sections focus on the individual subsystems of this architecture: Section 4 covers power and connectivity issues, Section 5 provides a rigorous analysis of the *Android Debug Bridge (ADB)* used by our SmartLab *Device Server (DS)* presented in Section 6. Section 7 summarizes our research and teaching activities using SmartLab, Section 8 enumerates our ongoing and future developments while Section 9 concludes the paper.

2 Related Work

This section provides a concise overview of the related work. SmartLab has been inspired by *PlanetLab* [30] and *Emulab* [17], both of which have pioneered global research networks; *MoteLab* [37], 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.

2.1 Remote Monitoring Solutions

There are currently a variety of Remote Monitoring Solutions (RMSs), including *Nagios* [26], a leading open-source RMS for over a decade, the Akamai Query System [9], STORM [14] and RedAlert [34]. All of these systems are mainly geared towards providing solutions for web-oriented services and servers. Moreover, none of those RMSs provide any tools related to the configuration and management of smartphone clusters. SmartLab focuses on providing a remote monitoring solution specifically for a smartphone-oriented cloud.

2.2 Wireless Sensor Network Testbeds

MoteLab [37] is a Web-based sensor network testbed deployed at Harvard University that has pioneered sensor network research. *CitySense [27]* has been MoteLab's successor enabling city-scale sensor network deployments. *Mobiscope [1]* is a federation of distributed mobile sensors into a taskable sensing system that achieves high density sampling coverage over a wide area through mobility. EU's *WISEBED* project [11] also federated different types of wireless sensor networks. Microsoft has made several attempts in building Sensor Networks with mobile phones [18], but none of these efforts has focused on smartphones in particular and their intrinsic characteristics like screen capturing, interactivity and power.

2.3 Smartphone Testbeds

There are currently several commercial platforms providing remote access to real smartphones, including *Samsung's Remote Test Lab [33], PerfectoMobile [29], Device Anyware [19] and AT&T ARO [3]*. 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 developed and launched *MegaDroid [36]*, 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.

2.4 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 [13]*, *CrowdLab [12]* and *PhoneLab [4]*, 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 [28] 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 micro-benchmarking results for a variety of modules that could be exploited by these systems.

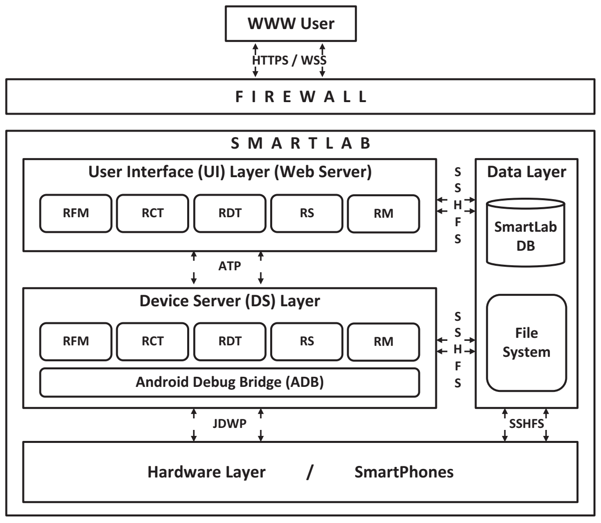


Figure 2: **The components of the SmartLab Architecture: We have implemented an array of modules wrapped around standard software to bring forward a comprehensive smartphone testbed management platform.**

3 SmartLab Architecture

In this section, we overview the architecture of our testbed starting out from the user interface and data layer moving on to the device server layer and concluding with the hardware layer, as illustrated in Figure 2. We conclude with an overview of our security measures and design principles.

3.1 User Interface and Data Layers

Interaction Modes: *SmartLab* implements several modes of user interaction with connected devices (see Figure 2, top-left layer) using either *Websocket-based interactions* for high-rate utilities or *AJAX-based interactions* for low-rate utilities. In particular, SmartLab supports: i) *Remote File Management (RFM)*, an AJAX-based terminal that allows users to push and pull files to the devices; ii) *Remote Control Terminals (RCT)*, a Websocket-based remote screen terminal that mimics touchscreen clicks and gestures but also enables users recording automation scripts for repetitive tasks; iii) *Remote Debug Tools (RDT)*, a Websocket-based debugging extension to the information available through the Android Debug Bridge (ADB); iv) *Remote Shells (RS)*, a Websocket-based shell enabling a wide variety of UNIX commands issued to the Android Linux kernels of allocated devices; v) *Remote Mockups (RM)*, a Websocket-based mockup subsystem for feeding ARDs and AVDs with GPS or sensor data traces encoded in XML for trace-driven experimentation.

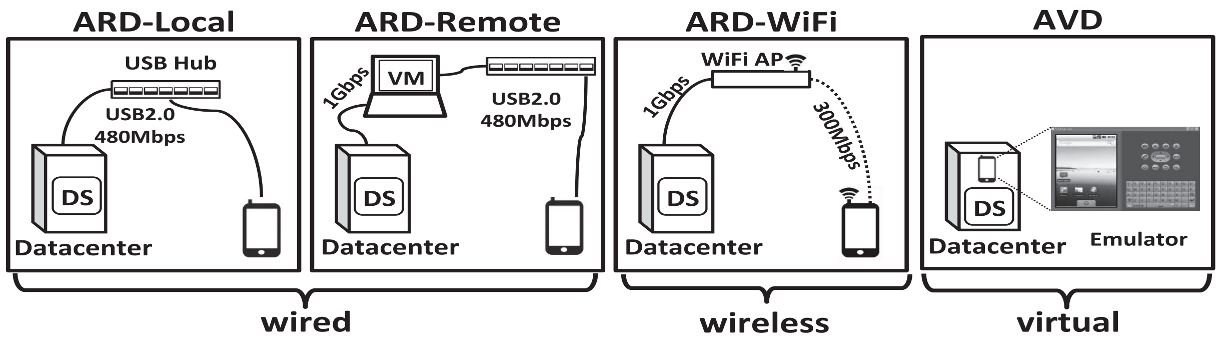


Figure 3: **Connection Modalities supported by SmartLab. ARD-Local**: Android Real Device (ARD) mounted locally to the Device Server (DS) through USB; **ARD-Remote**: ARD mounted through a USB port on a gateway PC to DS through a wired network; **ARD-WiFi**: ARD connected to DS through a WiFi AP; and **AVD**: Android Virtual Device running on DS.

WebSockets/HTML5: In order to establish fast and reliable communication between the *User Interface* and the underlying *Device Server (DS)*, SmartLab has adopted the HTML5 / WebSockets (RFC6455) standard thus enabling bi-directional and full-duplex communication over a single TCP socket from within the web browser. WebSockets are preferable for high-rate HTTP interactions, necessary in certain SmartLab subsystems, as opposed to AJAX calls that are translated into individual HTTP requests. WebSockets comprise of two parts: *i) an HTTP handshake*, during which certain application-level protocol keys are exchanged; and *ii) a data transfer phase*, during which data frames can be sent back and forth in full-duplex mode.

Currently, there are different types of Websocket handshakes implemented by web browsers (e.g., Hixie-75, 76, 00 and HyBi-00, 07 and 10). In order to support websockets on as many browsers as possible, we have modified an open-source plugin, the kanaka websockify plugin[4] formerly known as wsproxy, part of the noVNC project. 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.

File System: SmartLab currently utilizes a standard `ext4` local file system on the webserver. Upon user registration, we automatically and securely create a `/user` directory on the webserver with a given quota. Our filesystem is mounted with `sshfs` to all *DS* images running in our testbed, enabling in that way a unified view of what belongs to a user. In respect to the connectivity between the filesystems of smartphones, we currently support two different options: i) mounting the `/user` directory on the devices with sshfs; and ii) copying data from/to the devices through *ADB*. The former option is good for performance reasons, but it is only available on Android 4.0 ICS, which provides in-kernel support for user-space filesystems (i.e., FUSE). On the contrary, the latter option is more universal, as it can operate off-the-shelf and this will be the major focus in this work.

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

SSHFS & MySQL: Communication between the web server file system and the device server's remote file system is transparently enabled through the SSHFS protocol. The same protocol can also be utilized for offering a networked file system to the smartphones, as this will be explained later in Section 6. Finally, the web server also hosts a conventional MySQL 5.5 database utilized for storing data related SmartLab users, devices and remote device servers.

3.2 Device Server (DS) Layer

Overview: *DS* is the complete Linux OS image having the SmartLab subsystems and ADB installed, which connects an ARD or AVD to our *User Interface (UI)*. Currently, we are using CentOS 6.3 x64 with 4x2.4GHz virtual CPUs, 8GB RAM, 80GB hard disk for our images. User interface requests made through Websockets reach DS at a Prethreaded Java TCP Server with *Non-blocking I/O* and *logging*.

ATP: We have implemented a lightweight protocol on top of websockets, coined *ATP* (*ADB Tunnel Protocol*) for ease of exposition, in order to communicate *DS* data to the *UI* and vice-versa (see Figure 2). *Downward ATP* requests (from UI to DS) are translated into respective calls using the *ddmlib.jar* library (including `AndroidDebugBridge`) for file transfers, screen capture, etc., as well as *monkeyrunners* and *chimpchat.jar* for disseminating events (e.g., UI clicks). Alternatively, one *Downward ATP* request might also yield a stream of *Upward ATP* responses, as this is the case in our screen capturing subsystem (i.e., one screenshot request yields a stream of images) presented in Section 6.2.

Device Plug-n-Play: Physically connecting and disconnecting smartphones from DS should update the respective UI status as well. Consequently, we've exploited the respective `AndroidDebugBridge` interface listeners, issuing the SQL statements to our MySQL database and updating the device status changes on our website.

DS Limitations: Currently, each *DS* can only support up to 16 AVDs and theoretically up to 127 ARDs, due to limitations in the ADB server that will be presented in Section 5. In order to support a larger number of connected devices with the current ADB release, we utilize multiple-*DS*s 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.

DS Administration: In addition to the custom made Java Server each *DS* is also equipped with Apache and PHP. The local web server is responsible to host the administrative tools required for maintenance purposes similarly to routers and printers.

3.3 Hardware Layer

Hardware & OS Mix: SmartLab's hardware comprises both of *Android Smartphones* and 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. We additionally deploy over 40 Android smartphones and tablets from a variety of vendors (i.e., HTC, Samsung, Google, Motorola and Nokia). The majority of our smartphones came with pre-installed Android 2.1-2.3 (Eclair, Froyo, Gingerbread). These devices were "rooted" (i.e., the process of obtaining root access) and upgraded to Android 4.0.4 (Ice Cream Sandwich), using a custom XDA-Developers ROM, when their warranty expired. Notice that warranty and rooting are claimed to be irrelevant in Europe[5].

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.

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

Physical Connections: We support a variety of connection modalities (see Figure 3) that are extensively evaluated in Sections 4 and 5. In particular, most of our devices are connected to the server in ARD-Local mode, utilizing USB hubs, as this is explained in Section 4. Similarly, more smartphones are also connected from within our research lab, in the same building, using the ARD-Remote mode.

This mode is particularly promising for scenarios we want to scale our testbed outside the Department (e.g., *ARD-Internet mode*, where latencies span beyond 100ms), which will be investigated in the future. Finally, a few devices within the Department are also connected in ARD-WiFi mode, but additional devices in this mode can be connected by users as well.

3.4 Security Measures

Security is obviously a very challenging task in an environment where high degrees of flexibility to users are aimed to be provided. In this section, we provide a concise summary of how security is provided in our current environment.

Network & Communication: SmartLab DS-servers and smartphones are located in a DMZ to thwart the spread of possible security breaches from the Internet to the intranet. Although nodes in our subnet can reach the public Internet with no outbound traffic filtering, inbound traffic to smartphones is blocked by our firewall. Interactions between the user and our Web/DS servers are carried out over standard HTTPS/WSS (Secure Websockets) channels. DS-to-Smartphone communication is carried out over USB (wired) or alternatively over secured WiFi (wireless), so we increase isolation between users and the risk of sniffing the network.

Authentication & Traceability: Each smartphone connects to the departmental WiFi using designated credentials and WPA2/Enterprise. These are recorded in our SQL database along with other logging data (e.g., IP, session) to allow our administrators tracing users acting beyond the agreed "Use Policy".

Compromise & Recovery: We apply a resetting procedure every time a user releases a device. The resetting procedure essentially installs a new SmartLab-configured ROM to clear settings, data and possible malware/ROMs installed by prior users. Additionally, our DS-resident home directory is regularly backed up to prevent accidental deletion of files. Finally, users have the choice to shred their SDCard-resident data.

3.5 Design Methodology/Principles

SmartLab's architecture focuses on a number of desiderata such as *modularity, openness, scalability and expandability.* Its design was developed using a *"greedy"* bottom-up approach; in each layer/step, all alternative options that were available at the time it was designed were taken into consideration and the most efficient one was implemented. This was primarily because the research community long craved for the ability to test applications on real smartphone devices at the time SmartLab was designed. Because of this, we believed that there was no abundant time to dedicate for designing a *clean slate* architecture, like PlanetLab [30] and other similar testbeds. Additionally, some of the software/hardware technologies pre-existed in the laboratory and there was limited budget for upgrades. However, careful consideration was taken for each design choice to provide flexibility in accommodating the rapid evolution of smartphone hardware/software technologies.

4 Power and Connectivity

In this section, we present the bottom layer of the SmartLab architecture, which was overviewed in Section 3.3, dealing with power and connectivity issues of devices. In particular, we will analyze separately how *wireless* and *wired* devices are connected to our architecture using a microbenchmark that provides an insight into the expected performance of each connection modality.

4.1 Wired Devices

SmartLab *wired* devices (i.e., ARD-Local and ARD-Remote) are powered and connected through D-Link DUB-H7 7x port USB 2.0 hubs inter-connected in a *cascading* manner (i.e., "daisy chaining"), through standard 1.8mm USB 2.0 A-connectors rated at 1500mA. One significant advantage of daisy chaining is that it allows overcoming the limited number of physical USB ports on the host connecting the smartphones, reaching theoretically up-to 127 devices.

On the other hand, this limits data transfer rates (i.e., 1.5 Mbps, 480 Mbps and 5 Gbps for USB 1.0, 2.0 and 3.0, respectively). D-Link DUB-H7 USB 2.0 hubs were selected initially because they guarantee a supply of 500mA current on every port at a reasonable price, unlike most USB hubs available on the market. At the time of acquisition though, we were not sure about the exact incurred workloads and USB 3.0 hubs were not available on the market either.

USB 3.0: SmartLab is soon to be upgraded with USB 3.0 hubs that will support higher data transfer rates than USB 2.0. This is very important as in the experiments of Section 6.2, we have discovered that applications requiring the transfer of large files are severely hampered by the bandwidth limitation of USB 2.0 hubs (max. 480 Mbps). We have already observed that newer hubs on the market are offering dedicated fast-charging ports (i.e., 2x ports at 1.2A per port and 5x standard ports at 500mA per port) in order to support more energy demanding devices such as tablets.

Power-Boosting: Instead of connecting 6x devices plus 1x allocated for the next hub in the chain, we have decided to use 3x Y-shaped USB cables in our release. This allows ARDs to consume energy from two USB ports simultaneously (i.e., 2x500mA), similarly to high-speed external disks, ensuring that the energy replenishment ratio of smartphones will not become negative (i.e., battery drain) when performing heavy load experiments such as stress testing or benchmarks (e.g., AnTuTu) on certain Tablets (e.g., Galaxy Tab drew up to 1.3A in our tests). A negative replenishment ratio might introduce an erratic behavior of the smartphone unit, failure to function, or overloading/damaging the ports.

Power Profiling: In order to measure physical power parameters in our experiments, we employed the Plogg smart meter plug connected to the USB hub, which transmits power measurements (i.e., Watts, kWh Generated, kWh Consumed, Frequency, RMS Voltage, RMS Current, Reactive Power, VARh Generated, VARh Consumed, and Phase Angle) over ZigBee to the *DS*. These measurements are provided on-demand to the *DS* administrator through the Administrative Tools subsystem. Additionally, we have installed a USB Voltage/Ampere meter (see Figure 1 top-left showing 4.67V), offering on-site runtime power measurements of running applications.

4.2 Wireless Devices

In our current setup, *wireless* devices (i.e., ARD-WiFi) are operated by the SmartLab research team that powers the devices when discharged. Additionally, users can connect their own device remotely and these will be privately available to them only (e.g., see Figure 14 center). This particular feature is expected to allow us offering a truly programmable wireless fleet in the near future, as this is explained in Section 8. In this subsection, we will overview the underlying logistics involved in getting a wireless device connected to SmartLab over wireless ADB. Note that this process is automated through the SmartLab UI. In particular, the following commands have to be issued on rooted devices such that a smartphone can accept commands from the device server:

```
# On Smartphone (rooted):
# Enable ADB over wireless
#(to disable set port -1):
```

```
setprop service.adb.tcp.port 5555
stop adbd
start adbd
# On PC:
adb connect <device-ip>:5555
```

4.3 Connectivity Microbenchmark

In order to evaluate the time-efficiency of various connection modalities (i.e., wired or wireless) to our *DS*, we have performed a microbenchmark 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*.

Those experiments were conducted for calculating the time needed for transferring 2.5MBs to up to 16 devices. As we already mentioned in Section 3.2, those results can be generalized to larger configurations by increasing the number of *DS* images. 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 1: Transferring a 2.5MB file to 16 Devices

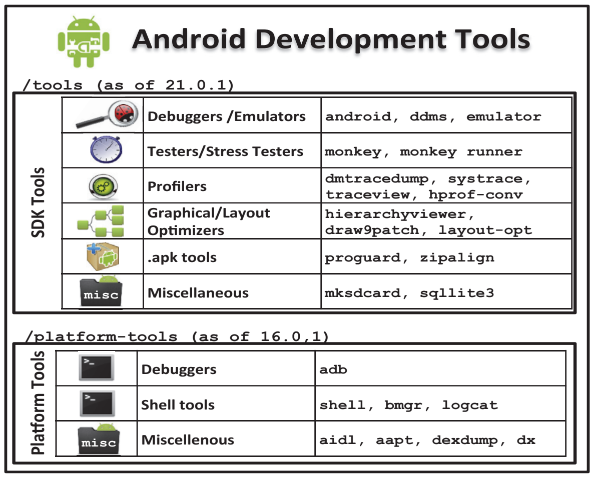
Connectivity Mode	Average Time (10 trials)
ARD-Local	1.4 seconds
ARD-Remote	4.8 seconds
ARD-WiFi	12 seconds

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. Yet, ARD-Remote devices are particularly promising for scaling our testbed outside the server room, thus are considered in this study.

5 Android Debug Bridge (ADB)

In this section, we provide an in-depth understanding of the *Android Debug Bridge* (ADB), which handles the bulk of communication between the connected smartphones and the *Device Server (DS)* (see Figure 2).

The `ADB` command (version 1.0.31, in this study) is part of the platform tools (version 16.0.1, in this study), provided by the Android development tools enabling the development, deployment and testing of applications using ARDs and AVDs. These tools are classified into two categories (see Figure 4): i) *the SDK tools*, which are platform-independent; and ii) *the Platform tools*, which are customized to support the features of the latest Android platform.


Android Development Tools

/tools (as of 21.0.1)

SDK Tools	Debuggers /Emulators	android, ddms, emulator
	Testers/Stress Testers	monkey, monkey runner
	Profilers	dmtracedump, systrace, traceview, hprof-conv
	Graphical/Layout Optimizers	hierarchyviewer, draw9patch, layout-opt
	.apk tools	proguard, zipalign
	Miscellaneous	mksdcard, sqllite3

/platform-tools (as of 16.0,1)

Platform Tools	Debuggers	adb
	Shell tools	shell, bmgr, logcat
	Miscellenous	aidl, aapt, dexdump, dx

Figure 4: **The Android Development Tools.**

In the latter category, there are also some shell tools that can be accessed through `ADB`, such as `bmgr`, which enables interaction with the backup manager of an Android device, and `logcat`, which provides a mechanism for collecting and viewing system debug output. Additionally, there platform tools such as `aidl`, `aapt`, `dexdump`, and `dx` that are typically called by the Android build tools or Android development tools.

5.1 Debugging Android Applications

Android applications can be developed using any Android-compatible IDE (e.g., Eclipse, IntellijIDEA, Android Studio) and their code is written using the JAVA-based Android SDK. These are then converted from Java Virtual Machine-compatible (`.class`) files (i.e., bytecode) to *Dalvik-compatible Executables* (`.dex`) files using the `dx` platform tool, shrinked and obfuscated using the `proguard` tool and ported to the device using the `adb install` command of ADB. The compact `.dex` format is specifically tailored for systems that are constrained in terms of memory and processor speed.

As illustrated in Figure 5 (right), each running application is encapsulated in its own process and executed in its own virtual machine (DalvikVM). Additionally, each DalvikVM exposes a single unique port ranging from 8600-8699 to debugging processes running on both local and remote development workstations through the *ADB*

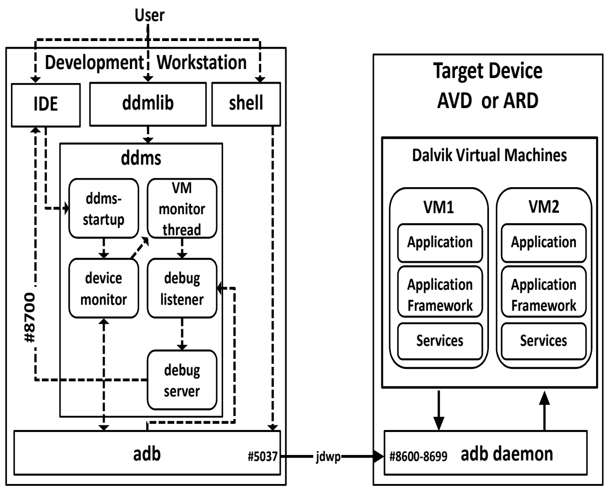


Figure 5: **Android Debug Bridge (ADB).** Overview of components involved in the debugging/deployment process of Android applications.

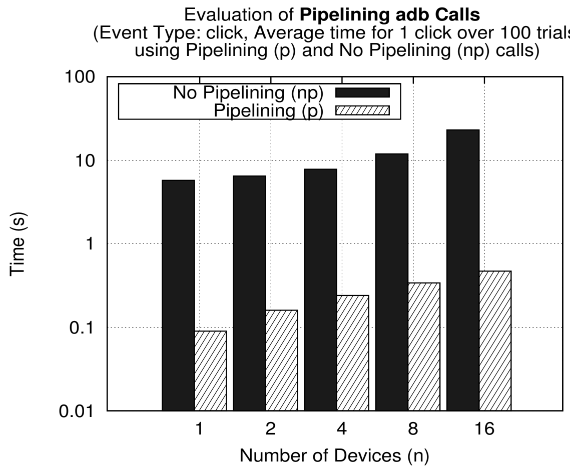


Figure 6: **ADB Pipelining Microbenchmark.** Evaluating the average time for one click with *pipelining* and *no-pipelining*. SmartLab utilizes *pipelining*.

daemon (adbd) shown in Figure 5 (left). The adbd is running on each target device and facilitates the connection with server processes (i.e., stream sockets) through the *java debug wire protocol* (jdwp-adb) transport protocol.

Debugging applications can be accomplished through the *Dalvik Debug Monitor Server* (DDMS), or its superset tool Android Device Monitor. DDMS can be executed as: i) a stand-alone application (ddms.jar); ii) initiated by the IDE; or iii) embedded to a java application (ddmlib.jar). All communication between DDMS and target devices is handled via ADB, which deploys a background process on the development machine, i.e., the *ADB server* that manages communication between the ADB client and the ADB daemon.

5.2 ADB Pipelining Microbenchmark

As mentioned in the previous section, users must connect directly on ADB or through a mediator library such as `ddmlib`, `monkeyrunner` or `chimpchat` both of which connect to ADB, in order to perform any action on a target device. However, initiating individual ADB connections for each action introduces a significant time overhead as it involves scanning for existing connections or creating new connections each time.

In order to justify this, we have conducted a microbenchmark using the Android chimpchat SDK tool, which allows amongst other functionality propagating events (e.g., mouse clicks) to a target device. More specifically, we generate 100 mouse click events and distribute them up to 16 ARD-Locals using two different settings: i) a new connection is initiated for each ADB call, denoted as *No Pipelining* (`np`); and ii) a single persistent connection is utilized for pipelining all ADB calls, denoted as *Pipelining* (`p`). The latter can be accomplished through the creation of a connection at the start of the script and then utilizing that connection for all propagated events. Note that the reason we have selected mouse click events is because they are extremely lightweight and do not introduce other time-demanding overheads (e.g., I/O), thus allowing us to focus on the time-overhead incurred by each connection when pipelining ADB calls or not.

Figure 6 shows the results of our evaluation (averaged over 100 trials). We observe that the overhead of not pipelining ADB calls is extremely high. In particular, 1 click on 16 AVDs using no-pipelining requires 23s, as opposed to pipelining that only requires 0.47s (i.e., a 98% improvement.) Such extreme time overheads may be a prohibiting factor for some applications, thus careful consideration must be taken to ensure that applications communicate with target devices through a single connection.

In SmartLab, we utilize a single persistent ADB connection for each particular component (e.g., separate ADB for Screen Capture and Shell Commands.) Through the persistent connection, all ADB calls are pipelined thus alleviating the aforementioned inefficiency. The above configuration offloads the issue of concurrent ADB calls to a single device from different components, to ADB and the OS, as a device is allocated to only one user at-a-time (thus high concurrency patterns are not an issue.)

6 Device Server (DS)

In this section, we present the middle layer of the SmartLab architecture, which was overviewed in Section 3.2 and illustrated in Figure 2, dealing with device management. In particular, we will explain and evaluate the following subsystems: *Filesystem and File Management, Screen Capture, Logging, Shell Commands* and *Sensor/GPS Mockups*.

6.1 File Management (RFM) Subsystem

We start out with a description of the File Management UI and finally present some performance microbenchmarks for pushing a file and installing an application on a device using ADB pipelining.

Remote File Management (RFM) UI: We have constructed an intutitive HTML5/AJAX-based web interface, which enables the management of the local filesystems on smartphones *individually* but also *concurrently* (see Figure 7). 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 7 (top-center). The user is then able to move a single file or multiple files to multiple target devices.

The 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.

File-Push Microbenchmark: The time required to transfer files from and to target devices differs significantly according to the type of device. In order to investigate this, we have conducted a microbenchmark that measures the time overhead for transferring files to/from the aforementioned different types of target de-

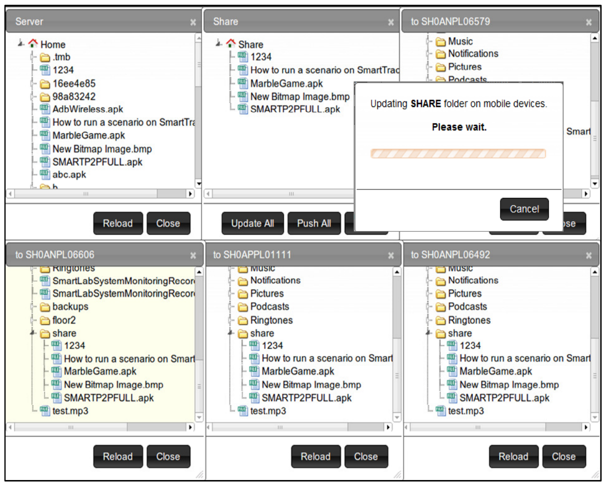


Figure 7: **Remote File Management (RFM) UI.** A share folder enables to push/pull files to devices concurrently. FUSE-enabled devices can feature sshfs shares.

vices. More specifically, we have utilized a 10MB file and distributed this file to up to 16 AVDs, ARD-WiFi, ARD-Remote and ARD-Local, respectively. 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.

The results are shown on the left side of Figure 8, which clearly illustrates 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.

File-Install Microbenchmark: In order to examine the cost of installing applications, which include transferring the application file (.apk) and its installation, we have conducted another microbenchmark 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 8. 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

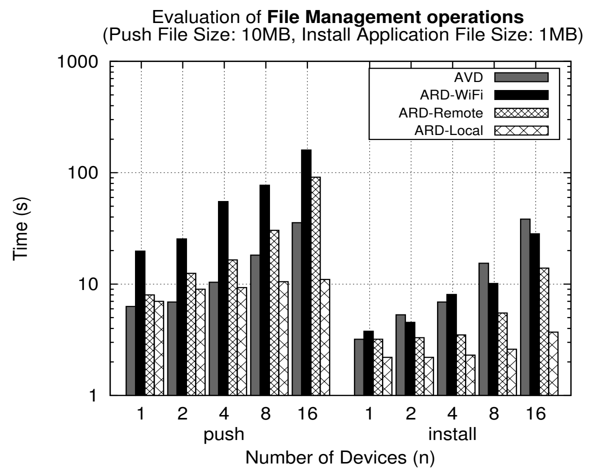


Figure 8: **File Management Microbenchmark.** Evaluating the average time for transferring files and installing applications on different types of target devices.

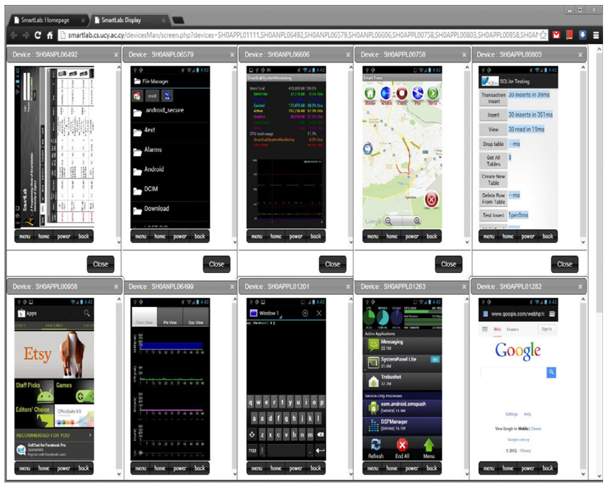


Figure 9: **Remote Control Terminal (RCT) UI.** Our implementation allows concurrent interaction (e.g., clicks, drag gestures, key press) on multiple devices.

from which 0.7s accounts for file transfer and 1.5s for installing the application. The results provide a clear indication that emulated devices 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.

6.2 Screen Capture (RCT) Subsystem

The Screen Capture subsystem enables capturing the actual screen of a target device so that it can be displayed to the user through the *Remote Control Terminal (RCT)* UI component (see Figure 9). Additionally, it supports a variety of events using the *chimpchat library* such as: i) *control events* (e.g., power button, home button); ii) *mouse events* (e.g., click, drag); and iii) *keyboard events* (e.g., key press).

Screen-Capture Alternatives: Capturing a screenshot of an ARD or AVD can be accomplished through the following means (either directly or through an application): i) on ARDs using the `cat` command (`/dev/fb0` or `dev/graphics/fb0` according to the target device version) and redirecting the output to an image file; ii) on both using the Android `monkeyrunner` script command `takeSnapshot()`; iii) on both by continuously invoking the `getScreenShot()` command provided by the ddmlib library; and iv) on both similarly to (iii), by continuously listening to the direct stream that contains the contents of each consecutive screenshot (i.e., `readAdbChannel()` in ddmlib). The SmartLab screen capture component has been developed using the (iv) approach because it is more efficient both in terms of memory and time as it utilizes buffered-oriented, non-blocking I/O that is more suitable for accessing and transferring large data files as shown next.

Screen-Capture Microbenchmarks: In order to justify our selection, we have performed a microbenchmark that evaluates the time required to generate and transfer 100 consecutive screenshots from up to 16 ARD-Local devices using the (ii) and (iv) approaches denoted as *monkeyrunner python scripts* and *Screen Capture*, respectively. Approaches (i) and (iii) were omitted from the experiment because the former cannot provide a continuous stream of screenshots required by RCT and the latter does not provide any guarantee that a screenshot image will be ready when the ddmlib library's `getScreenShot()` command is invoked, which may lead to presentation inconsistencies. The experiment was performed only on ARD-Local devices that outperform AVD, ARD-Remote and ARD-WiFi devices w.r.t. file transfer operations as is the case of capturing and displaying a screenshot image.

The results of our microbenchmark, depicted in Figure 10 (left), clearly justify our selection. In particular, SmartLab's Screen Capture subsystem always maintains a competitive advantage over monkeyrunner python scripts for all number of target devices. Additionally, we notice that the time required for processing images for up to 8 devices is almost identical at 0.97±0.03s. However, when 16 devices are used, the time required

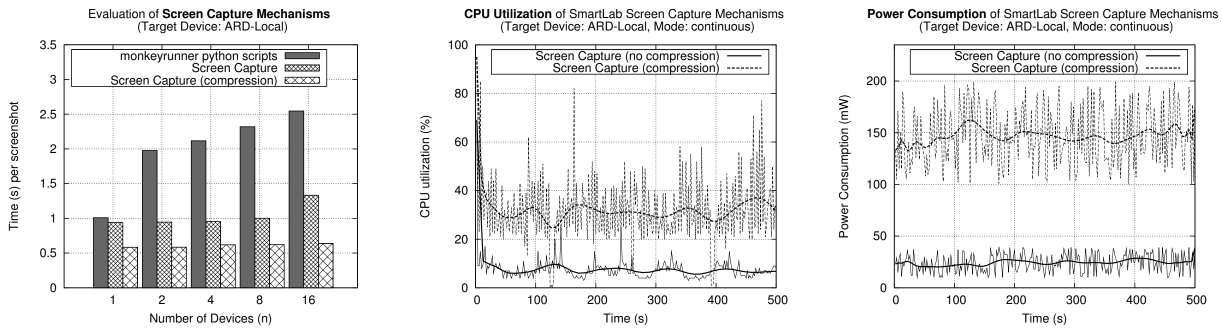


Figure 10: **Screen Capture Subsystem Microbenchmarks** on ARD-Local devices. **(left)** Evaluating the time overhead of capturing screenshot images using monkeyrunner python scripts and SmartLab's Screen Capture subsystem; Evaluation of SmartLab Screen Capture compression mechanism w.r.t.: **(center)** CPU Utilization.; and **(right)** Power Consumption.

for processing the screenshot images increases by ≈30% (i.e., 1.33s±0.6s). This may be inefficient in the case of applications requiring real-time control of the target devices. On the other hand, for automated bulk tests the above is not a big problem, as these are not affected by the interaction latency that is inherent in any type of remote terminal like ours. By performing a number of file transfer benchmarks on our USB 2.0 hubs, we discovered that this happens because the maximum available bandwidth for file transfer was approximately 250Mbps (theoretically up-to 480Mbps). Consequently, it was not adequate to support the necessary 320Mbps USB bandwidth incurred by the 16 devices each transferring a 480x800 screenshot with an approximate size of 2.5MB per shot (i.e., 16 x 2.5MB x 8bps = 320Mbps).

On-Device Compression: Currently, producing large screenshot files cannot be avoided as there are no mechanisms for reducing the file size (i.e., compression). In order to alleviate this problem, we experimented with an in-house module for rooted devices that provides the ability to generate compressed screenshot images (e.g., JPEG, PNG) locally at the device prior to transmitting them over the network. We evaluated the revised Screen Capture subsystem, denoted *Screen Capture (compression)* using the same configuration as in the previous experiment.

We observe in Figure 10 (left) that the Screen Capture (compression) clearly outperforms the *Screen Capture (with no compression)*, as expected. This is because the files generated by Screen Capture (compression) never reached over 45KBs. As a result, the revised Screen Capture subsystem is not affected by the limitation of the USB 2.0 hub as the combined bandwidth rate required was 5.7Mbps (i.e., 16 x 45KB x 8bps) and this is the reason why the time required per screenshot for all number of devices remains persistent at 0.6±0.05s.

Power and CPU issues: Compressing images though, requires additional CPU effort as well as increased power consumption on a smartphone. In order to investigate these parameters, we have utilized a custom SmartLab System Monitor application (see Figure 9, third screenshot on top row for overview) and PowerTutor tools (see on the same figure the second screenshot on bottom row), in order to measure CPU utilization and power consumption, respectively. Our findings are illustrated in Figure 10 (center and right). We observe that the CPU utilization in the compression scenario reaches 28±15% as opposed to 7±3% when no compression is performed. This is important as applications requiring high CPU utilization should use the conventional (i.e., no compression) approach. Similarly, the power consumption of compression is higher. However, the difference is very low compared to other smartphone functions (e.g., 3G busy ≈ 900mW [8]). In the future, we aim to investigate automated techniques to switch between available screen capture modes.

6.3 Logging (RDT) Subsystem

The SmartLab Logging subsystem is responsible for parsing the debug data generated locally at each target device and providing comprehensive reports regarding the status of each target device to the user. The log data is generated automatically by the Android OS and includes various logs such as system data, system state and error logs. These can be accessed directly through the ADB commands `dumpsys`, `dumpstate`, and `logcat` respectively or through the `bugreport` command, which combines all previous logs into a comprehensive log file.

The Logging subsystem is accessible through the *Remote Debug Tools (RDT)* component of the web server. The logging process starts with the RDT component, which upon a user request for logs initiates a web request

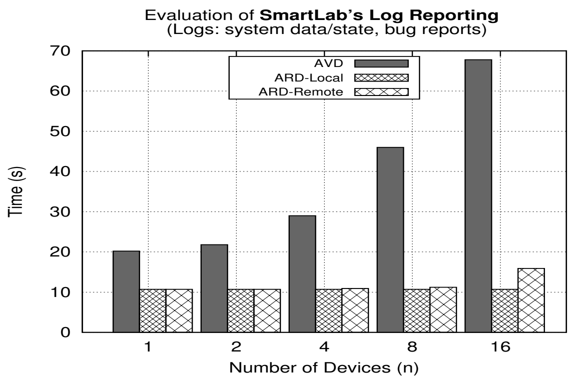


Figure 11: **Logging (RDT) Microbenchmark.** Time required for retrieving the report log from different target devices.

including the target device id (or multiple devices ids) using AJAX. The Logging subsystem receives this request and propagates an `adb bugreport` command to the target devices selected by the user. Consequently, the resulting log report is saved to a separate directory inside the user home directory and the user is able to choose whether to manually traverse the report or to use a more sophisticated tool such as the ChkBugReport tool [6] that is able to illustrate the results in a comprehensive graphical manner. If the latter is chosen, the Logging subsystem invokes the ChkBugReport tool, passing the log report file as a parameter. Finally, the resulting HTML files generated by the ChkBugReport tool are stored in the users' "Home" directory.

Currently, the Logging subsystem (using ChkBugReport) extracts the following information: *i) Stacktraces; ii) Logs; iii) Packages; iv) Processes; v) Battery statistics; vi) CPU Frequency statistics; vii) Raw data; and viii) Other data.* Additionally, each ChkBugReport plugin can detect (possible) errors, which are highlighted in the errors section of the HTML report files. For instance, by looking at the Stack-trace section the user might observe deadlocks or strict mode violations in addition to other useful information.

We have conducted a microbenchmark in order to evaluate the time overhead for gathering log reports from the target devices. More specifically, we gathered the bugreports from up to 16 AVDs, ARD-Remote and ARD-Local devices, respectively. The results, shown in Figure 11 clearly illustrate that ARD-Remote and ARD-Local devices outperform AVDs. This confirms again that utilizing real devices can speed up the experimental process and produce output results more efficiently.

[6]Check Bug Report, *http://goo.gl/lRPUW*.

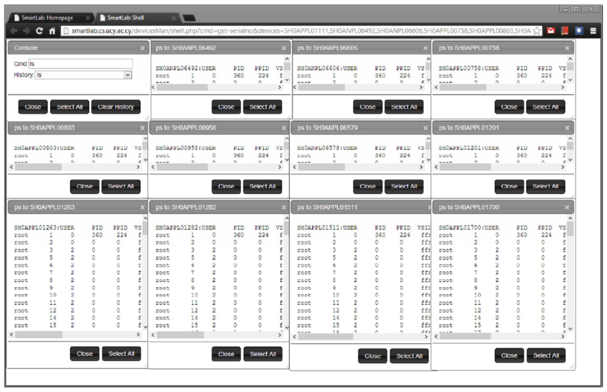


Figure 12: **Remote Shell (RS) UI.** Allows concurrent UNIX command executions on multiple devices.

6.4 Shell Commands (RS) Subsystem

The Shell Command subsystem works in collaboration with the web server's *Remote Shells (RS)* component (see Figure 12) in order to execute shell commands from SmartLab to all target devices selected by the user. These commands include every available `adb shell` command supported by the Android OS on rooted devices and a considerable subset on non-rooted devices. The RS component propagates each shell command through a bi-directional WebSocket to the Shell Commands subsystem, which in turn executes each command on the target devices and returns the resulting data back to the RS web interface. More specifically, it allows a user to select a set of target devices and launch a separate window consisting of frames (JQuery dialogs) for each target device. Each frame represents an interactive shell on the selected device allowing the user to remotely issue shell commands to single or multiple devices at the same time.

6.5 Sensor/GPS Mockup (RM) Subsystem

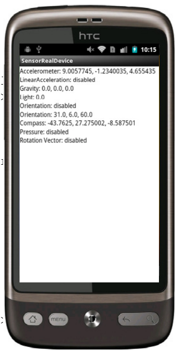
A mockup provides part of a system's functionality enabling testing of a design. In the context of Android, Mockup refers to the process of extending an AVD's or ARD's particular sensor or GPS with custom values. Additionally, one important benefit of Mockups is that these can support the addition of sensors that may not exist in the hardware of a particular ARD (e.g., NFC). The Android SDK 4.0 supports the mockup of GPS data through the following command sequence:

```
# On PC running AVD (5554: emulator)
telnet localhost 5554
geo fix latitude longitude
```

In order to support both GPS and other sensor mockups in SmartLab, (e.g., *accelerometer, compass, orienta-*

```
<state>
  <sensor type="proximity">
    <value>0.0</value>
  </sensor>
  <sensor type="linear
            acceleration">
    <value>0.0</value>
    <value>0.0</value>
    <value>0.0</value>
  </sensor>
  <sensor type="orientation">
    <value>31.0</value>
    <value>6.0</value>
    <value>60.0</value>
  </sensor>
  <sensor type="pressure">
    <value>0.0</value>
  </sensor>
  <sensor type="gyroscope">
    <value>0.0</value>
    <value>0.0</value>
    <value>0.0</value>
  </sensor>
  <sensor type="rotation
                 vector">
    <value>0.0</value>
```

```
    <value>0.0</value>
    <value>0.0</value>
  </sensor>
  <sensor type="light">
    <value>0.0</value>
  </sensor>
  <sensor type="magnetic fie
    <value>-43.7625.0</value
    <value>27.275002</value>
    <value>-8.587501</value>
  </sensor>
  <sensor type="accelerometer
    <value>9.0057745</value>
    <value>-1.23400</value>
    <value>4.655435</value>
  </sensor>
  <sensor type="gravity">
    <value>0.0</value>
    <value>0.0</value>
    <value>0.0</value>
  </sensor>
  <sensor type="temperature"
    <value>0.0</value>
  </sensor>
</state>
```



Accelerometer: 9.0057745, -1.2340035, 4.655435
LinearAcceleration: disabled
Gravity: 0.0, 0.0, 0.0
Light: 0.0
Orientation: disabled
Orientation: 31.0, 6.0, 60.0
Compass: -43.7625, 27.275002, -8.587501
Pressure: disabled
Rotation Vector: disabled

Figure 13: **Sensor/GPS Mockup (RM): (left, center)** A data trace of various sensor measurements encoded in XML. The given file can be loaded to ARDs and AVDs through this subsystem; **(right)** An application built with SLSensorManager using the measurements.

tion, temperature, light, proximity, pressure, gravity, linear acceleration, rotation vector and gyroscope sensors) on both ARDs and AVDs, we opted for a custom module.

In particular, we have been inspired by the *SensorSimulator*[7] open source project, which establishes a socket server on *DS* feeding devices with sensor or GPS readings encoded in XML (see Figure 13 left). As this functionality is completely outside the ADB interaction stream, we were required to provide each application with a custom library, coined `SLSensorManager.jar`.

SLSensorManager Library: Our library can be embedded to any Android application enabling interaction with the SmartLab GPS/Sensor subsystem running on *DS*. For example, Figure 13 (right) shows how a sample application has been constructed with this library. In fact, our library has precisely the same interface with the Android SDK SensorManager, consequently a user can override Android's default behavior very easily achieving in that way to feed its allocated device from a real and realistic sensor dataset.

Hardware Emulation: 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.

[7]Openintents, `http://goo.gl/WkuN`

7 Experiences using SmartLab

In this section, we present four different research efforts, including: GPS-trajectory benchmarking [38], peer-to-peer search [22], indoor positioning [24] and database benchmarking (the last carried out in the context of a graduate course.) None of the following studies would have been feasible with *AVDs*, as all of the below scenarios require *fine-grained* and *low-level* access (e.g., sdcard, WiFi, real CPU and mobility).

7.1 Trajectory Benchmarking

SmartLab has been utilized in the context of the SmartTrace[38] project[8], which is a prototype crowd-sourced trajectory similarity search framework enabling analytic queries over outdoor GPS traces and indoor WiFi traces, stored on users' smartphones.

SmartLab was utilized to carry out a GPS mockup study with the GeoLife GPS Trajectories [39]. The SmartLab file management functionality was extremely useful in disseminating input traces and collecting our experimental results from the local sdcards of smartphones. Additionally, the Remote Control Terminals were equally important in order to setup and run the experiments. Finally the device diversity allowed us to test trajectory comparison algorithms on many smartphones (see Figure 14, left).

As SmartLab is currently firewalled (i.e., the device server is only accessible through the webserver), it is not feasible to have some outside process connect to the SmartLab smartphone processes internally. In order to overcome this correct security configuration, we wrote our Smarttrace smartphone clients in a manner that these only issued outgoing TCP traffic (i.e., connecting to the outside server) as opposed to incoming TCP traffic.

Finally, in order to scale our experiments to 200 smartphone processes, in the absence of such a large number, we launched 10 concurrent threads to each of our 20 reserved ARD devices.

7.2 Peer-to-Peer Benchmarking

SmartLab was also utilized in the context of a Peer-to-Peer benchmarking study (i.e., the SmartP2P [22] project). SmartP2P offers high-performance search and data sharing over a crowd of mobile users participating in a social network. Similarly to SmartTrace, the experimental evaluation of SmartP2P was performed on real devices reserved through SmartLab. A subtle difference of this study was that the UI interactions were recorded from within RCT into automation scripts stored on SmartLab. Those scripts, running on our Device

[8]SmartTrace, `http://smarttrace.cs.ucy.ac.cy/`

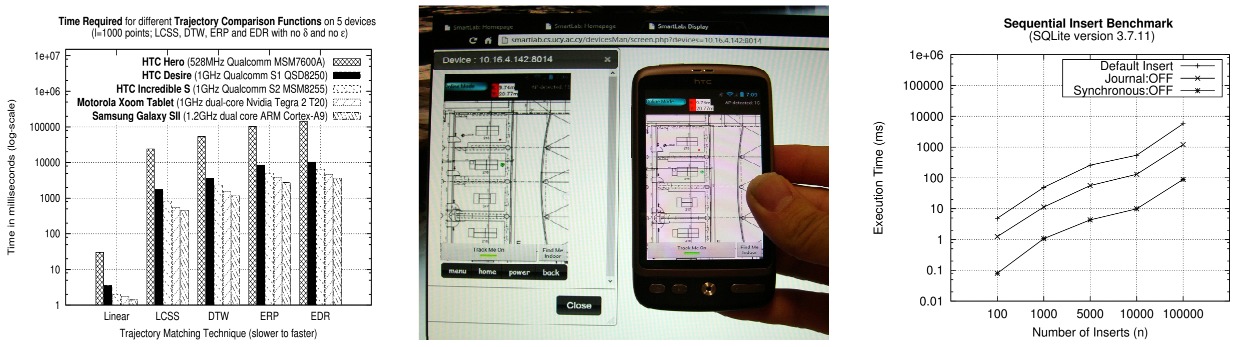


Figure 14: **Research with SmartLab.** (**Left**) Testing trajectory comparison algorithms on a diverse set of smartphones in SmartTrace [38]; (**Center**) Testing indoor localization using ARD-WiFi mode in Airplace [24]; (**Right**) Testing various SQLite tuning parameters in the context of an advanced databases course.

Server, would automatically repeat an experimental simulation improving automation and repeatability of the experimental evaluation process.

7.3 Indoor Localization Testing

WiFi-based positioning systems have recently received considerable attention, both because GPS is unavailable in indoor spaces and consumes considerable energy. In [24], we have demonstrated an innovative indoor positioning platform, coined Airplace, in order to carry out fine-grained localization with WiFi-based RadioMaps (i.e., 2-4 meters accuracy). SmartLab has facilitated the development, testing and demonstration of Airplace and its successor project Anyplace[9] considerably as explained next.

Firstly, we extensively used the ARD-WiFi mode, which allowed us to move around in a building localizing ourselves while exposing the smartphone screen on a remote web browser through SmartLab (e.g., see Figure 14, center). The particular setting has proved considerably useful for demonstrations 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.

Secondly, SmartLab allowed us to collect and compare *Received Signal Strength (RSS)* indicators from different WiFi chip-sets, which is important for RSS measurements and would not be possible with *AVDs*. Finally, SmartLab allowed us to test the generated APK on a variety of devices.

7.4 DB Benchmarking

A recent study by NEC Labs America [20], has shown that underlying flash storage on smartphones might be a bottleneck in many smartphone applications, which cache results locally.

In the context of an Advanced DB course at our department, students were asked to carry out an extensive experimental evaluation of SQLite, the most widely deployed SQL database engine in the world that is readily available by the Android SDK. One particular objective of this study was to find out how the reads and writes could be optimized. For the given task students parsed the sqlite data files stored by various smartphone apps in their sqlite dbs. Subsequently, students carried out a number of trace-driven experimentations.

Figure 14 (right) for example, shows how sequential inserts are affected by disabling the `PRAGMA synchronous` and `PRAGMA journal_mode` runtime options on a smartphone storing its data on a FAT32-formatted sdcard. In respect to SmartLab, it is important to mention that APK and data trace files were seamlessly transferred to target devices. Additionally, after installing the APKs it was very efficient working on several RCT control terminals concurrently, carrying out the experimental study quickly.

8 Future Developments

In this section, we outline some of our current and future development plans:

8.1 Experimental Repeatability

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 and academic and industrial research over the last decades. For instance, the TREC Conference series co-sponsored by National Institute of Standards and Tech-

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

nology (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.

In the context of our project we are: i) collecting our own data on campus (e.g., WiFi RSS data [24]) and additionally trying to convince other research groups contributing their own data to the SmartLab repository. In respect to storage, we are using a prototype Apache HBase installation within our datacenter, to store sensor readings in a tabular and scalable (i.e., column-oriented) format.

Apache HBase is an open-source version of Google's Bigtable [7] work utilized to store and process Crawling data, Maps data, etc., without the typical ACID guarantees that are slowing and scaling down distributed relational databases (e.g., MySQL-Cluster-like DBs). The given store can be utilized to store billions of sensor readings that can be advantageous to our *GPS/Sensor Mockup* subsystem. This will allow a researcher to test an algorithm or application using tens or hundreds of smartphone devices using automated scripts, similarly to [36] but with bigger data. Another envisioned scenario would be to enable smartphone experimentation repeatability and standardization.

8.2 Urban-scale Deployment

We are currently working with local telecommunication authorities in order to obtain mobile data time for our mobile fleet and local transportation companies in order to have them move our devices around in a city, with possible free WiFi access to their customers as an incentive.

The envisioned scenario here is to be able to test an algorithm, protocol or application with ARD-Mobile devices in an urban environment, providing in that way an open mobile programming cloud. This could, for example, support data collection scenarios, e.g., *VTrack [35], CitySense [27], and others,* which rely on proprietary software/hardware configurations, but also online traffic prediction scenarios, trajectory and sensor analytics, crowdsourcing scenarios, etc.

Such ARD-Mobile devices need of course limiting the capabilities of users (e.g., prohibit the installation of custom ROMs, disable camera, sound and microphone.) We are addressing this with a customized after-market firmware distribution for Android (i.e., ROM), named CyanogenMod [10]. We did not opt for the *Android Open Source Project (AOSP)*, as it was fundamentally difficult to port the drivers of all ARD we have ourselves. Moreover, notice that the AOSP project currently supports only the Google Nexus family [11] of phones off-the-shelf. Enabling urban sensing scenarios also has a legal dimension as Europe has a strict Data Protection Policy (e.g., Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data.)

[10]CyanogenMod, `http://www.cyanogenmod.org/`

8.3 Web 2.0 API

We are currently working on a Web 2.0 JSON-based API of our testbed using the Django framework[12]. Django comes with rich features including a *Model-View-Controller (MVC)* architecture that separates the representation of information from the users' interaction with it. In particular, this effort will allow users to access the subsystems of our testbed in a programmable manner (i.e., Web 2.0 JSON interactions) and write applications to extend SmartLab, similarly to NagMQ [32].

Consider for instance the Eclipse IDE, which we are currently extending with Smartlab integration functionality through its API. The high level idea here is to allow developers to compile their code and deploy it immediately on available devices accessible on SmartLab, by having the Smartlab UI become part of the Eclipse IDE.

Finally, we are considering the integration with Google App Inventor[13], such that programmers can see their developments immediately on SmartLab.

8.4 Federation Issues and PG Management

Our Web 2.0 API will allow us to implement SmartLab federation scenarios. For example, groups around the globe can interface with SmartLab enabling a truly global smartphone programming cloud infrastructure. Additionally, we aim to develop a SmartLab derivative for *Personal Gadget (PG)* management, which was motivated in the introduction. This will be facilitated by the fact that personal gadgets are quantitatively and qualitatively increasing but more importantly, by the fact that PGs are *reusable* after they become deprecated as they are *programmable* and *feature-rich*.

8.5 Security Studies

SmartLab can be utilized in order to conduct experiments related to enhanced smartphone security and privacy. SmartLab smartphones can be used as honey pots for investigating intruders' behavior. Additionally, smartphones can be used as replicas of real devices enabling

[11]Nexus Factory Images, `http://goo.gl/v1Jwd`
[12]Django Framework, `https://www.djangoproject.com/`
[13]MIT AppInventor, `http://appinventor.mit.edu/`

the replication of real event execution performed on real devices. As a result, researchers can use SmartLab in order to identify newly introduced threats by gathering statistics from multiple replicas. Furthermore, SmartLab can be utilized in the context of projects using replicated execution [31] for validation and verification purposes. Carefully investigating the security aspects related to SmartLab will be a topic of future research. At the end, SmartLab's administrators will be able to present their experiences related to managing security in a distributed mobile environment similarly to the work presented by Intel on how to secure PlanetLab [6].

9 Conclusions

In this paper, we have presented the first comprehensive architecture for managing a cluster of both real and virtual Android smartphones. We cover in detail the subsystems of our architecture and present micro-benchmarks for most of the internally components.

Our findings have helped us enormously in improving the performance and robustness of our testbed. In particular, by pipelining Android Debug Bridge (ADB) calls we managed to improve performance by 98%. Additionally, by compressing screen capture images with moderate CPU overhead we improve capturing performance and minimize the network overhead.

This paper has also presented four different research and teaching efforts using SmartLab, including: GPS-trajectory benchmarking, peer-to-peer search, indoor positioning and database benchmarking. Finally, this paper has overviewed our ongoing and future SmartLab developments ranging from Web 2.0 extensions to urban-scale deployment primitives and security.

Our long-term goal is to extend our testbed by engaging the research community that can envision and realize systems-oriented research on large-scale smartphone allocations but also enable a platform for *Personal Gadget (PG)* management.

Acknowledgments

We would like to thank Matt Welsh (Google) and Stavros Harizopoulos (HP Labs) for the useful discussions that lead to the realization of this work. Finally, we would like to thank our USENIX LISA'13 shepherds, Carolyn Rowland and Adele Shakal, and the anonymous reviewers for their insightful comments. This work was financially supported by the last author's startup grant, funded by the University of Cyprus. It has also been supported by EU's COST Action IC903 (MOVE), by EU's FP7 MODAP project and EU's FP7 Planetdata NoE.

References

[1] Tarek Abdelzaher, Yaw Anokwa, Peter Boda, Jeff Burke, Deborah Estrin, Leonidas Guibas, Aman Kansal, Sam Madden, and Jim Reich. *"Mobiscopes for Human Spaces"*, IEEE Pervasive Computing, Volume 6, Issue 2, April 2007.

[2] David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. *"FAWN: A fast array of wimpy nodes"*, In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (*SOSP'09*). ACM, New York, NY, USA, 1-14, 2009.

[3] AT&T Application Resource Optimizer (ARO), Free Diagnostic Tool: `http://goo.gl/FZnXS`

[4] 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)"*, 9th USENIX conference on Networked systems design & implementation (*NSDI'12*). USENIX Association, Berkeley, CA, USA, 2012.

[5] Jeffrey Bickford, H. Andrs Lagar-Cavilla, Alexander Varshavsky, Vinod Ganapathy, and Liviu Iftode. *"Security versus energy tradeoffs in host-based mobile malware detection"*, In Proceedings of the 9th international conference on Mobile systems, applications, and services (*MobiSys'11*). ACM, New York, NY, USA, 225-238, 2011.

[6] Paul Brett, Mic Bowman, Jeff Sedayao, Robert Adams, Rob Knauerhase, and Aaron Klingaman. *"Securing the PlanetLab Distributed Testbed: How to Manage Security in an Environment with No Firewalls, with All Users Having Root, and No Direct Physical Control of Any System"*, In Proceedings of the 18th USENIX conference on System administration (*LISA'04*). USENIX Association, Berkeley, CA, USA, 195-202, 2004.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. *"Bigtable: a distributed storage system for structured data"*, In Proceedings of the 7th symposium on Operating systems design and implementation (*OSDI'06*). USENIX Association, Berkeley, CA, USA, 205-218, 2006.

[8] Georgios Chatzimilioudis, Andreas Konstantinidis, Christos Laoudias, and Demetrios Zeinalipour-

Yazti. *"Crowdsourcing with smartphones"*, In IEEE Internet Computing, Volume 16, 36-44, 2012.

[9] Jeff Cohen, Thomas Repantis, Sean McDermott, Scott Smith, and Joel Wein. *"Keeping track of 70,000+ servers: the akamai query system"*, In Proceedings of the 24th international conference on Large installation system administration (*LISA'10*). USENIX Association, Berkeley, CA, USA, 1-13, 2010.

[10] Cory Cornelius, Apu Kapadia, David Kotz, Dan Peebles, Minho Shin, and Nikos Triandopoulos. *"Anonysense: privacy-aware people-centric sensing"*, In Proceedings of the 6th international conference on Mobile systems, applications, and services (*MobiSys'08*). ACM, New York, NY, USA, 211-224, 2008.

[11] Geoff Coulson, Barry Porter, Ioannis Chatzigiannakis, Christos Koninis, Stefan Fischer, Dennis Pfisterer, Daniel Bimschas, Torsten Braun, Philipp Hurni, Markus Anwander, Gerald Wagenknecht, Sndor P. Fekete, Alexander Krller, and Tobias Baumgartner. *"Flexible experimentation in wireless sensor networks"*, In Communications of the ACM, Volume 55, Issue 1, 82-90, 2012.

[12] 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.

[13] 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.

[14] M. Dehus, and D. Grunwald. *"STORM: simple tool for resource management"*, In Proceedings of the 22nd conference on Large installation system administration conference (*LISA'08*). USENIX Association, Berkeley, CA, USA, 109-119, 2008.

[15] Jakob Eriksson, Lewis Girod, Bret Hull, Ryan Newton, Samuel Madden, and Hari Balakrishnan. *"The pothole patrol: using a mobile sensor network for road surface monitoring"*, In Proceedings of the 6th international conference on Mobile systems, applications, and services (*MobiSys'08*). ACM, New York, NY, USA, 29-39, 2008.

[16] 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.

[17] 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.

[18] Aman Kansal, Michel Goraczko, and Feng Zhao. *"Building a sensor network of mobile phones"*, In Proceedings of the 6th international conference on Information processing in sensor networks (*IPSN'07*). ACM, New York, NY, USA, 547-548, 2007.

[19] Keynote Systems Inc., Device Anywhere: `http://goo.gl/mCxFt`

[20] 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.

[21] 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.

[22] Andreas Konstantinidis, Demetrios Zeinalipour-Yazti, Panayiotis G. Andreou, Panos K. Chrysanthis, and George Samaras. *"Intelligent Search in Social Communities of Smartphone Users"*, In Distributed and Parallel Databases, Volume 31, 115-149, 2013.

[23] Emmanouil Koukoumidis, Li-Shiuan Peh, and Margaret Rose Martonosi. *"SignalGuru: leveraging mobile phones for collaborative traffic signal schedule advisory"*, In Proceedings of the 9th international conference on Mobile systems, applications, and services (*MobiSys'11*). ACM, New York, NY, USA, 127-140, 2011.

[24] 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.

[25] Kaisen Lin, Aman Kansal, Dimitrios Lymberopoulos, and Feng Zhao. *"Energy-accuracy trade-off for continuous mobile device location"*, In Proceedings of the 8th international conference on Mobile systems, applications, and services (*MobiSys'10*). ACM, New York, NY, USA, 285-298, 2010.

[26] Nagios Enterprises LLC., `http://www.nagios.org/`.

[27] Rohan Narayana Murty, Geoffrey Mainland, Ian Rose, Atanu Roy Chowdhury, Abhimanyu Gosain, Josh Bers, and Matt Welsh. *"CitySense: An Urban-Scale Wireless Sensor Network and Testbed"*, In Proceedings of the 2008 IEEE Conference on Technologies for Homeland Security (*HST'08*), IEEE Computer Society, Washington, DC, USA, 583-588, 2008.

[28] Adam J. Oliner, Anand P. Iyer, Eemil Lagerspetz, Ion Stoica and Sasu Tarkoma. *"Carat: Collaborative Energy Bug Detection (poster and demo)"*, 9th USENIX conference on Networked systems design & implementation (*NSDI'12*). USENIX Association, Berkeley, CA, USA, 2012.

[29] Perfecto Mobile, `http://goo.gl/DSlP9`.

[30] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. *"A Blueprint for Introducing Disruptive Technology into the Internet"*, A blueprint for introducing disruptive technology into the Internet. SIGCOMM Comput. Commun. Rev. 33, 1 (January 2003), 59-64, 2003.

[31] 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.

[32] Jonathan Reams. *"Extensible Monitoring with Nagios and Messaging Middleware"*, In Proceedings of the 26th international conference on Large Installation System Administration: strategies, tools, and techniques (*LISA'12*). USENIX Association, Berkeley, CA, USA, 153-162, 2012.

[33] Samsung, Remote Test Lab: `http://goo.gl/p7SNU`

[34] Eric Sorenson and Strata Rose Chalup. *"RedAlert: A Scalable System for Application Monitoring"*, In Proceedings of the 13th USENIX conference on System administration (*LISA'99*). USENIX Association, Berkeley, CA, USA, 21-34, 1999.

[35] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson. *"Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones"*, In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (*SenSys'09*). ACM, New York, NY, USA, 85-98, 2009.

[36] Tim Verry. *"MegaDroid simulates network of 300,000 Android smartphones"*, Extremetech.com, Oct 3, 2012. `http://goo.gl/jMaS8`.

[37] 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.

[38] Demetrios Zeinalipour-Yazti, Christos Laoudias, Costantinos Costa, Michalis Vlachos, Maria I. Andreou, and Dimitrios Gunopulos. *"Crowdsourced Trajectory Similarity with Smartphones"*, IEEE Trans. on Knowl. and Data Eng. 25, 6 (June 2013), 1240-1253, 2013.

[39] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. *"Mining interesting locations and travel sequences from GPS trajectories"*, In Proceedings of the 18th international conference on World wide web (*WWW'09*). ACM, New York, NY, USA, 791-800, 2009.