\documentclass[conference,letterpaper]{IEEEtran}

\usepackage[english]{babel}

\usepackage[dvips]{graphicx}

\usepackage{graphicx}

\usepackage{longtable}

\graphicspath{{./figs/}}

%%\usepackage[T1]{fontenc}

%%\usepackage[latin2]{inputenc}

%%\usepackage{t1enc}

\usepackage{epsfig}

\begin{document}

\selectlanguage{english}

\title{DroidLab: Mobile Sensing Made Easy, Fast and Cheap}

\author{\IEEEauthorblockN{Bal\'azs Lajtha, Rolland Vida}

\IEEEauthorblockA{Department of Telecommunications and Media Informatics\\

Budapest University of Technology and Economics\\

% Magyar tud\'osok k\"or\'utja 2., Budapest, Hungary, H-1117 \\

Email: \{lajtha.balazs, vida\}@tmit.bme.hu}}

%\maketitle

% \long\def\symbolfootnotetext[#1]#2{\begingroup \def\thefootnote{\fnsymbol{footnote}}\footnotetext[#1]{#2}\endgroup}

% \symbolfootnotetext[0]{The second author was supported by the Janos Bolyai

% Fellowship of the Hungarian Academy of Sciences.}

\maketitle

\begin{abstract}

We are living in a parallelized world. What was thought to be impossible for one can now easily be achieved by many. On one hand, we have the cloud: an army of obeying faceless machines. On the other hand we have the crowd: A colorful collection of volunteers. Clouds are built and sold but to control the crowd, each member of has to be won over.

\end{abstract}

\begin{IEEEkeywords}

\end{IEEEkeywords}

\section{Introduction}

\label{sec:intro}

"Crowdsourcing is the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call." This was the initial definition of crowd-sourcing when first used in 2006 by Jeff Howe and Mark Robinson. Two main orthogonal problem sets can be solved using crowd-sourcing: data gathering and data processing. When a data gathering problem is distributed among members of the crowd we use the term crowd sensing or participatory sensing, while for distributed computational tasks we use the general term crowd sourcing.

Another way to categorize crowd-sourced applications is whether the contributors profit directly of the gathered or processed information. Wikipedia uses the power of the crowd to provide a service that can directly be used by the contributors. When it comes to crowd sourcing platforms, creating a framework to acquire information is much easier than creating a generic framework that is able to present users with the results. In many cases the beneficiaries of the crowd sourced tasks are different than those providing the data. DroidLab targets these scenarios.

Probably the most popular crowd sourced computational problem was SETI@home, a project lasting several years and aimed to search for extraterrestrial intelligence by analyzing radio waves.. The problem was divided into a large number of subtasks and distributed among the willing participants. Users could set a SETI screen-saver that downloaded and solved these subtasks when the user was away. SETI's success came from it's innovative design and from the interesting task that it performed. From the point of view of the volunteer, this was a low cost, low gain activity. PCs back than had long startup times and high standby power consumption, so once SETI was installed it didn't generate a visible overhead, and provided the user with an insight of the performed work. Some early gamification elements, like leaderboards, were also implemented.

A somewhat different example is the Google Image classification game, in which two players cooperated to solve a simple puzzle. An image was displayed in both players' browsers. The first player had to find five words that described the image. Then, the second player had to guess those words. This human image processing game allowed Google to improve Image Search results. The players were motivated to play the game for the game itself, and for added gamification elements.

Both examples involved utilization of computational resources owned by the users to perform a highly parallelized task that the users perceived as common good, but they didn't benefit directly from the results of their work. In both cases, users received an input that they processed based on a given algorithm. The tasks were not personalized. SETI@home's tasks were solely performed by a machine, while Google Image games had to be solved by human users.

Participatory sensing on the other hand involves distributed data collection . Sensing can be automated, or can involve the active participation of the user. Before smartphones, sensing was usually done through mostly fixed sensors that were deployed to monitor a given phenomenon at a specific location. They were not able to move, and they had very restricted computational, memory and communication resources, so as to spare their very limited batteries. In case those batteries were depleted, sensors had to be collected, recharged and redeployed, which was a very cumbersome, and sometimes even impossible task. The evolution of smartphones opened the way for many new complex sensing applications. Smartphones have the connectivity that enables real time, high bandwidth communication. The devices can access precise location data, and are equipped with a wide range of sensors from cameras and microphones to accelerometers, gyroscopes and barometers. New mobile operating systems offer many seamless ways for the sensing system to prompt for user input. In addition, smartphones are owned by individuals who recharge them regularly, so energy efficiency becomes less critical.

Given these possibilities, a large number of smartphone-based crowd-sensing applications emerged recently. However, on one hand, all these applications have several common parts (e.g., communication with the sensor, data collection, data sending, or data processing modules) which are written from scratch by each application developer. In addition, if two or more such applications run on the same phone, they will all consume memory and CPU resources in a redundant and unnecessary way. On the other hand, once the application is written, a large “crowd” of users is needed to test, run and use this application. Gathering such a crowd is however a difficult task, as the users should be motivated in a way to upload and run these specific applications on their personal devices.

. DroidLab is being developed to tackle exactly these issues. It is a cloud-backed mobile framework that aims to solve key challenges of mobile crowd sourcing. First, it provides the research community and the mobile application developers with an easy-to-use skeleton on which they can readily build their own crowd-sourcing applications, benefiting from the provided sensing and communication modules. Second, the DroidLab framework will run on a large number of smartphones, enabling application developers to upload their own code on these devices and test it on a large-scale testbed.

DroidLab is being developed as part of the EIT ICT Labs FITTING European research project, which aims to federate several large scale wired and wireless testbeds, such as PlanetLab, OneLab or SensLab. The goal of the project is to provide a common interface and a transparent access to this large set of heterogeneous resources, enabling researchers to run experiments and test their various protocols and applications. DroidLab adds to FITTING the ability to support mobile nodes and mobility scenarios, a feature that was not addressed by the other, previously mentioned testbeds.

The rest of this paper is organized as follows. First, as related work, in section II we present some promising mobile crowd sourcing applications and frameworks. In section III we identify the key challenges of mobile crowd sourcing and define the use cases for our DroidLab framework. Then, in section IV we present the architecture and design of our solution and a short evaluation of the current framework. Finally, in section V we conclude the paper and propose some future works.

\section{Related work}

\label{sec:related\_work}

We already mentioned two successful examples of crowd sourcing, SETI@home and Google Image Game, which both targeted desktop platforms. Seti@Home was an automated solution while Google Image Game involved user interaction.

While it's easy to find proposed applications for participatory sensing, and academic papers discussing the challenges of such applications and their effects on society, there is a very limited umber of successfully deployed applications. In 2012 the mobile broadband penetration was 74.8% in the developed world, but only 19.8% in the developing world. While this equals in a larger number of subscribers in the developing regions, most application downloads come from the developed countries. Hence, most crowd sourced applications address "first world problems". Here is a short survey of the tackled scenarios, and a brief presentation of some existing crowd-sourcing frameworks.

\subsection{Crowd-sensing application scenarios}

Transportation is an area where several crowd-sensing applications emerged recently. When using a car, knowing the traffic conditions in advance gives a huge advantage. Waze is one of the many traffic aware navigation applications that uses smartphone sensors to identify traffic conditions. Besides traffic, road quality is an important factor too, both for drivers and cyclists. Smartphone acceleration sensors can accurately detect potholes. While no single pothole detection application has emerged, several publications discuss such a framework.

When arriving to the destination finding a parking spot will become the main concern. Google launched Open Spot, an experimental application that relied on user interaction to detect freed up parking spaces. The application was a failure, and the problem remains unsolved. Finally, public transportation can also benefit of crowd sensed information. HotStop Live is for example an application that provides live transit information gathered by the users.

Presence and phone book applications can also benefit of crowd-sensing.The user's presence can be detected using sensor information, enabling a framework to communicate detailed information about a contact's state before calling him/her. Phone books can be (partially) shared among users to ease reverse lookup of phone numbers. Mr Number helps its users to avoid unwanted calls by collaboratively maintaining a black list of telemarketers and robocallers.

Weather and environment monitoring is probably the most popular crowd-sensing scenario. We participated for example in the development of the mobile application Id\"ok\'ep, a Hungarian weather service. Id\"ok\'ep had more than a hundred deployed online weather stations and the mobile application enabled users to share weather photos, perceived weather conditions and analogue measurement results.

While smartphones are not equipped with the proper sensors to become a weather station, their near field communication technologies (Bluetooth, WiFi, NFC) enable them to be extended with peripherals capable of measuring air quality, humidity or temperature. In the Common Sense project Bluetooth enabled air pollution sensors were deployed with the mobile application to measure and report air pollutants. Microphones enable smart phones to capture noise level and process noise pollution.

Finally, CreekWatch is an application that helps to monitor water quality and pollution relying on user reports and photos.

Monitoring network coverage and connection quality can also be done through crowd-sensing. The two most important communication channels for smartphones are WiFi networks and cellular networks. WiFi Finder builds and maintains a database of open WiFi networks and their properties. RootMetrics on the other hand monitors cellular coverage, as it lets users share their active measurement data to process a coverage map.

Lifestyle and health

Healthcare in modern societies is a main concern. Smartphones' sensors can help on many different levels. For individuals it can help to monitor activity, track exercises, with external Bluetooth devices log blood pressure and other metrics. In a family results of the elderly members can trigger alarms at the younger generations. Same can be implemented in a smaller community. On national scale data provides insight of the health level of the people. On a global scale these statistics can help find correlation between lifestyle and health.

A crowd sourced ad-hoc communication network can be useful both on outdoor events where cellular coverage is not designed for many users, and in case of an outage to help first responders contact those without mobile service. Crowd sensing is useful to detect and track parades, estimate number of participants.

\subsection{Crowd-sourcing Frameworks}

Several frameworks have been proposed for different aspects of mobile crowd sourcing, most of them being experimental. mCrowd is a mobile crowdsourcing platform that enables iPhone users to post or tag images or answer questions. mCrowd was a prototype application that demonstrated a micro-payment system rewarding the contributors.

mClerk is a platform that distributes tasks and enables workers to post their responses. mClerk targeted third world users without smartphones. The key challenge of mClerk was to create a communication protocol that enables data transfer over text messages, enabling traditional phone users to receive complex tasks and report results. Both platforms focus on the active contribution of the users.

Other frameworks target the communication protocols used in crowd sourcing applications. When information is both gathered and consumed by the same crowd, it is crucial to have a robust, multi-source, multi-destination multicast infrastructure. Fan-Bai et al proposed such a system for vehicular networks. Demirbas et al developed an overlay that uses Twitter to distribute sensor information. Szabo et al investigated the use of XMPP as a carrier protocol for crowdsourced applications.

Haderer et al presented the design of a crowd sensing framework called APISENSE.

In [] they identify the main challenges that they face. Their solution, a multi-cloud based SsaS platform focuses on the serving architecture.

They promote a multi-tenant approach to increase security, privacy and robustness. APISENSE is composed of a central cloud component and of discrete sensing nodes, each running in the cloud. Sensing tasks are defined using their domain specific language. These definitions are then processed by a Software Production Line that creates the cloud based sensing nodes. Device users are then prompted to join these sensing nodes. APISENSE takes a cloud centric approach, focusing on the gathering, storage and processing of the received data. Aiming to solve the same problem, with DroidLab we take a more client centric approach.

Our solution is designed to comply with Android application guidelines. By adjusting DroidLab to the Android eco system, we believe that it will be more appealing and more trustworthy to the users, resulting in higher level of engagement.

MEDUSA is also an experimental framework that enables the execution of sensing tasks on a swarm of devices connected to a cloud based server. Ryong Ra et al created a scripting language MedScript and a prototype of the runtime. MedScript introduced an abstraction that helps to describe crowdsourcing tasks along with language constructs that map to user involvement, like requesting user input or user actions and rewarding users. MEDUSA also defines a software architecture and provides a reference implementation for cloud-backed crowd sourcing frameworks.

As opposed to MEDUSA, DroidLab focuses on automated sensing. We designed our framework to be able to operate in the background, but taking in consideration the user's resources. Our first contribution is a detailed permission and quota management system that gives full control to the users who run the DroidLab framework on their phones.

Our second contribution is the plugin-based design that allows the framework to be tailored to the capabilities of the specific device. There exist several thousand different makes of Android devices, dozens of sensors, several different network stack implementations with different feature sets, and many different types of usages. DroidLab contains different plugins for different sensors and data types. This enables the user to configure his DroidLab installation to his comfort zone. We took in account the permission system of Android, and designed the framework to require only network access in itself.

Our third contribution is the DroidLab API, a Java API that eases the development of crowd sourcing application with functions to utilize the plugins’ functionality, and gives access to sensing related tasks, like scheduling and data collection.

In the following section we present the intended use cases of the framework and we address the challenges of such a platform.

\section{Solution}

\label{sec:solution}

DroidLab is a cloud backed mobile framework. It consists of a cloud application running on Google App Engine. We will reference this app as the server. The mobile devices participating in DroidLab are called clients. Clients run the DroidLab framework which consists of the core and several plugins. The framework is capable of downloading and running crowd sourcing applications, called applications. To avoid confusion, users who upload and run their applications will be called Developers. Applications rely on events fired by the plugins and methods offered by the plugins to perform sensing tasks. The framework provides means to the applications for uploading results.

DroidLab is under development. The Android client is already functional, and most of the features mentioned in this paper are already available. Limited server functionality is currently provided by a PHP based portal.

\subsection{Use cases}

\label{sec:use\_cases}

Applications mentioned in the previous chapter have several properties in common:

- Applications only work in areas where the application's coverage is high

- They provide services to the users

- Services provided can be consumed on the same mobile device, through a specific application

For such an application to succeed the provided service has to be in balance with the required contribution, and the application has to be advertised and distributed fast. These requirements result in well polished, feature rich applications that emphasize the provided services, and high marketing costs. If an application doesn't reach enough users or active user contribution is required and is too costly (see Google's parking app) the service dies out. This is a high cost, high risk scenario.

DroidLab has three main use cases. It aims at bootstrapping such applications. If an application developer needs sensory data for the provided service to work, he can deploy a sensor application to DroidLab and gather the required information before the launch of the stand alone application. If a service requires mobile sensed data, but doesn't offer mobile content to the users, it is really hard to motivate users to download yet another application that will run in the background and consume valuable resources. If a device runs DroidLab, such applications can be deployed seamlessly, without distracting the user. Finally DroidLab's initial purpose was to be a research platform, a mobile test network similar to PlanetLab and SenseLab. It is especially difficult to deploy applications for research purposes, as they usually don't provide any value to the user, operate only for shorter periods of times and usually target specific user groups. DroidLab provides a solution for both use cases.

\subsection{Target group}

The majority of the currently sold phones are smartphones, regardless of gender, age and technical knowledge users are persuaded to buy smartphones, resulting in the wide penetration of these devices. We designed DroidLab to not require any maintenance. After the framework is set up on the device and configured properly for the users privacy requirements and available resources, it's self sustained, no user interaction is needed for it's operation. This enables the deployment of DroidLab on devices that are owned by users who are not familiar with modern technology. This widens the pool reachable users. With proper marketing and communication every smart phone user can be targeted, but the main target group of the DroidLab framework are students and young adults living in urban regions of first world countries, owning smartphones with mobile broadband access. They travel daily in their home town by different transport means, use multiple WiFi access points, and near field radio devices (Bluetooth, RFID, NFC).

\subsection{Challenges}

\subsubsection{Modularity}

DroidLab's Android framework from an Android perspective consists of a core application and plugin applications. Each plugin can be installed or removed as an Android application. Each plugin requires the permissions that are needed for it's use. This enables the user to assemble a setup that she is comfortable with, and is suitable for her device's capabilities. If she doesn't want to share her location, than omitting that plugin the framework will be limited on the OS level not to use the location services.

The clear benefits of this modular approach come with some drawbacks. The DroidLab distributions will be fragmented, devices will have different plugin sets and even plugin versions. Users will also have the possibility to revoke permissions from the framework and limit the resource usage of plugins.

To address this issue, DroidLab has dual conformance checking for the applications. When a Developer request devices for his application, he indicates the span of the deployment. He may target a specific user group - age, gender, country, urban area or countryside, device type, user activity type - and weather the application needs inter user connections or devices can be scattered over a larger area. He can request device capabilities: location sensor accuracy, network connectivity type, available sensors. And finally estimates the resources his application will consume on a device during the applications lifetime.

A strong point of DroidLab is the management of the resources. As different sensors have different types of resources, we introduced a resource description system. For each resource we define quotas. Quotas can be associated with the volume of consumed resources and the speed of consumption. A resource can have several associated quota. Quotas are used to present users with resource management decisions and to limit an application's resource usage.

A plugin interface is defined so that plugins can expose the quotas. Each quota has five usage levels. The user can choose not to grant any of that resource. Low, medium and high are different levels defined by the plugins and labeled by human readable labels. Finally unlimited indicates the user is willing to provide every available resource of the kind to the framework.

Applications can be present parallel on a device. The quota limits defined by the user apply for the aggregated resource consumption of the application. The server is in charge of distributing the applications among devices in a way that each application will be able to access the amount of resources the developer required. This is the first level of conformance.

Once an application is pushed to a device, each delivered event and called method is monitored by the framework and authorized by the resource manager running on the device. Once the application's quota level is reached or the overall quota limit has expired, the framework denies the request.

\subsubsection{Security}

DroidLab ensures that the user is protected from both outside attacks and malicious applications. We identified three high risk points in the architecture.

Framework and plugin integrity

We rely on Android Play Store's security measures to guarantee that nor the nor the framework or the plugins code can be compromised. Both the framework and the plugins can be downloaded from the Play Store through the Google Play Store application. This ensures that no third party application will be able to use the same package name on the device. Android user management system ensures code integrity: DroidLab isn't more fragile than any Android application.

Application download from the server

An attacker could compromise the DroidLab client by pushing an application to the client that either has altered quotas and permissions, or contains code that was not checked and compiled on the DroidLab server. To prevent such an intrusion, every application and accompanying resource descriptor is signed by the DroidLab's asymmetrical key. The application has the public key compiled inside itself. This signature is checked every time the application's is loaded from local storage.

Malicious applications

Applications are written in Java, compiled by the server and sent to the client. To make sure the client framework has full control over the application, only a limited set of the Java language can be used. Developers are limited to java.lang, java.util and java.math packages, excluding threading, Timer and class loading. Developers are allowed to use DroidLab's interfaces package but cannot reference any class from the framework. Source code is checked for these imports and compiled without any external library. This ensures that applications can not reach outside of the designated sandbox.

Inter-component communication

Our solution uses intents to communicate on the device between the different components. Android permissions are defined and enforced for these intents limiting senders and receivers of the intent to the trusted components of the client (framework and plugins). It is up to the user to keep his device safe, any installed Android application has to be authorized, permissions granted manually.

\subsubsection{Privacy}

One of the main user concerns is privacy. Such a framework can be successful only if the users can be certain that the data shared through the application doesn't violate their privacy. The required level of privacy varies from person to person, and DroidLab is designed to allow different levels of privacy.

The common basis of privacy is device anonymity. Each device generates a unique identifier and every data is associated to that identifier. Identifier is independent of any other device or user identifier. That means that if a user reinstalls the application, the two installations will have different identifiers and cannot be associated. As the framework only handles the task of storing log lines generated by the applications, other privacy issues are taken in account on the plugin level. We design every plugin with user's privacy in mind. Permissions defined by the plugins are designed in a way to accommodate different privacy needs.

Log files have to be stored on the external storage, as most current phones have limited internal storage available. These files can be read and altered by any application having permission to the SD card. To protect the sensitive information, we use a XOR symmetric key coding and sign the saved files. This guarantees integrity and privacy. Files will be uploaded through a secure channel either manually or automatically. In case of manual settings the user can review file's contents before deciding to delete or upload them.

Files uploaded to the server are stored in the user's private space. Users can access the information that will be shared by them. Three sharing options will be available to the user, always share will grant developers access to the files immediately, speeding up the data processing. Share if not removed will grant developers access the files after a given period if user doesn't delete then. The strictest option share when permitted keeps files private until the user doesn't give explicit permission to share the files with the developer. Files can be reviewed but not altered by the user.

\subsubsection{Seamlessness}

DroidLab will run on user equipment, tablets and phones that have to perform well even when DroidLab is running sensing applications. Moreover these devices are mostly mobile, with limited or costly network access.

Battery life

Our preliminary experiments show that the framework in itself doesn't influence battery life. We created three passive monitoring applications: a battery meter, a CPU utilization capturer and a running application capture, both scheduled to run each second. Running these application didn't increase the battery usage considerably, battery lifetime was not compromised.

Active sensing like gyroscope, accelerometer, WiFi or Bluetooth discovery, active bandwith measurements will naturally consume more battery. Plugins can be optimized to reduce power consumption. To prevent the framework from draining the battery, thresholds can be set by the user, when to disable battery intensive tasks, and when to stop the framework altogether.

Another battery intensive task is the upload of the periodic collected sensing data. As this is not a time sensitive task, the framework takes in consideration the user's network preference, and the user can indicate at what battery level and charging state the framework should upload gathered data.

Performance

Current Android distributions don't support Android application level CPU limits and quotas, so our options were limited in terms of processor usage management. As DroidLab application source code is checked to prevent starting new threads, the lifecycle of the application is managed by the framework. We are measuring the runtime of each application method call and flagging applications that take too long to return. However plugin methods initiated by the applications are asynchronous, hence we do not measure CPU load resulting from a plugin call directly. An estimation of the CPU usage of CPU intensive plugin method calls will be incorporated in the quotas defined by the plugins.

Our application design guidelines suggest a timer or event based operation, with small tasks that aim to filter sensor data or change the state of the application. DroidLab was not intended to be a distributed mobile computation platform.

Storage space

The Android platform distinguishes between internal storage and external storage. Internal storage is private to the Android application using it for downloaded DroidLab applications adds another layer of security. Available storage capacity varies widely from device to device. The usual bottleneck is internal storage. Our internal storage needs are way more modest than most Android applications. However caching application logs before uploading them may result in the overuse of external storage.

\subsubsection{Incentive}

Successful crowd sourced applications provide valued services to the users in return for their collaboration. Frameworks like DroidLab cannot provide such value to the users due to the diverse usage of the captured data. While a service using basic sensing (open WiFi map, broadband coverage map, DTN messaging service) could be provided, our aim is different. Another approach taken by proposed crowd sourcing frameworks is the introduction of micro-payment systems to incentivize users in participating. A micropayment system in an automated sensing scenario opens up many issues. As opposed to easy to evaluate tasks like answering a question or taking a picture at a given location, it is hard to put a price tag on a periodic or event based sensing task. A monetary reward would also attract cheaters.

We believe that other extrinsic motivators can be used to increase public involvement. We propose a gamified approach. By introducing game elements in the configuration and running of the application it would be more appealing to the target group.

Goal

The framework in itself is just a delivery system for applications. Our goal is to be able to run more applications on more devices, and to be able to target specific user groups. This requires that users install more plugins and give more permissions and higher quotas. And user should provide detailed user profiles of themselves.

On the other hand each application benefits from different user behaviors. A pothole detecting application needs car users. Drivers using roads that are not yet part of the database are more valuable than users taking the same path every day. For an application that maps opened WiFi networks, people not switching on their WiFi connection, or devices used only at home are useless.

We plan on implementing a system that has both a unified reward system and an application level reward system. The gamified system will be composed of points, leaderboards and badges. Points will be awarded for quotas that the applications spend. As developers have limited quota available, applications will be optimized to use fewer resources. So a quota spent by the application should be useful for the developer. For the users to increase the spent quota they have to make their setup desirable to the applications, meaning more plugins, permissions and higher quota limits. Leaderboars will be based on the points awarded to the users. With the increase of the user count, we will create different leaderboards for different time windows, and social leaderboards allowing users to compare themselves to their peers.

Badges will be the parts where the application developers can task and reward their users. Applications have the possibility to define badges that will be part of the system. Specifications of the task that has to be accomplished to earn the badge can be secret or known to the user. In the later case, the framework will present the user with the description of the task. The application will be responsible for checking weather the requirements have been fulfilled. The framework will also define badges. Framework provided badges will be part of the tutorial introducing the user to DroidLab. User tasks will guide the user through the settings, and give the user some insight about the system.

\subsubsection{Scaling}

As opposed to crowd sensing applications, DroidLab will not have bootstrapping issues. DroidLab doesn't provide a service that relies on sensing data, the first user will receive the same user experience as a user joining the system when it becomes popular. In fact, early adopters will benefit from their history when it'll come to leaderboards or badges: as badges are associated to applications, if an application is available through a limited time only, badges earned during that period will be more valuable. DroidLab's success won't depend on it's startup speed.

DroidLab's server requests needed for it's operation scale linearly with the number of devices. Most of the server interactions consist of uploading or downloading files related to a single user or a single application. Gamification related server requests will also involve a single user's information. Leaderboards will be cached and evaluated periodically. This makes the cloud an ideal platform for DroidLab. A more complex task is the distribution of applications among devices. Finding an optimal solution is a hard problem. We will use sub-optimal algorithms to find acceptable associations. This will result in sub-optimal device quota usage but will conserve server resources.

\section{Conclusions}

\label{sec:conclusion\_and\_future\_work}

\section{Acknowledgment}

\label{sec:acknowledgment}

http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a\#subscribers

http://au.businessinsider.com/comparing-app-downloads-across-countries-2013-6

http://www.androidauthority.com/google-labs-open-spot-a-useful-application-that-no-one-uses-15186/

http://www.waze.com/

http://live.hopstop.com/

http://www.bizjournals.com/sanjose/news/2013/05/31/whitepages-acquires-crowdsourced-spam.html

http://www.lvmt.fr/ewgt2012/compendium

http://www.communitysensing.org/

https://play.google.com/store/apps/details?id=com.jiwire.android.finder

http://www.rootmetrics.com/

\end{document}

%%% Local Variables:

%%% mode: latex

%%% TeX-master: t

%%% End: