PhoneLab Documentation

Release stable

PhoneLab Team

June 05, 2017

Contents

CHAPTER 1

What's New

- [2017-04-21] PhoneLab will be shut down on May 2017.
- [2016-09-14] PhoneLab has migrated to Nexus 6 devices. We are ready to accept experiment requests.

Overview

PhoneLab is a smartphone platform testbed based on Android. As a researcher, you can add instrumentations to learn how Android works in the wild, or make experimental changes to test your new ideas. Either way, PhoneLab provides a way to monitor, study and experiment on Android system at scale, with the power of full system control.

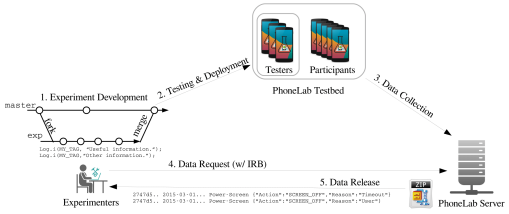


Fig. 1.1: Overview of PhoneLab Experiment Process.

More details on how to deploy experiments on PhoneLab can be found in rest of this documentation, but the high level steps are:

1. Contact us to state who you are, and what kind of experiments you want to do on PhoneLab. We do not have any specific application templates—just use common sense.
2. We will determine whether the experiment is suitable one PhoneLab. Typically we welcome ideas that requires changes at platform level, which is the unique capability of PhoneLab.
3. Upon approval, we will create a experiment branch for your on our Gerrit server so that you can download PhoneLab AOSP source and make changes.
4. When your changes are ready, we will merge them into our mainstream release branch and push out to participants.
5. Meanwhile, you will need to apply for **IRB approval** to obtain the experiment data generated by your changes.

Get Started

PhoneLab use the same tools to manage platform source with AOSP, except that we are hosting our own platform mirror. Here we are not trying to cover every aspect of the building process, which is already well documented by AOSP.

Registration

First, you will need to register an account on our Gerrit server at http://platform.phone-lab.org:8080. You will need to use the OpenID authentication provided by Yahoo!, since Google has terminated its OpenID support.

Then please sign in and fill up your account information, most notably your SSH public key and email address. These two information are required later on to clone the platform source.

Finally, please open the email sent by Gerrit to confirm your email address, and let us know your Gerrit account name.

Downloading and Building

Please follow the AOSP instructions to set up your local develop environment.

Before you continue, make sure that you have contacted us with these information:

- Who you are
- What the experiment is about (be brief)
- What will be a good code name (it will be used in creating the experiment branch `experiment/cm-13.0/${id}/${codename}`, where `${id}` is assigned by us.)
- Your account name and email on our Gerrit server.

Next, you are ready to clone the source code.

```
$ repo init -u ssh://<USERNAME>@platform.phone-lab.org:29418/cm-shamu/manifest -b <EXPERIMENT_BRANCH>
$ repo sync
```

Where `<USERNAME>` is your user name on our Gerrit server, and `<EXPERIMENT_BRANCH>` is the branch name we created for you.

Since `repo sync` will put every repository in a "detached head" mode, you may want to check out your experiment branch so further changes will be staged on your branch:

```
$ repo forall -pvec git fetch phonelab <EXPERIMENT_BRANCH>:<EXPERIMENT_BRANCH>
```

Now you can go a head and build the platform. Note the build target is for Nexus 6, aka "shamu".

```
$ source build/envsetup.sh
$ lunch cm_shamu
$ make -j 16
```

After the compilation finishes, you can use `fastboot` to flash the images to your device. Given that you are going to be modifying the platform we suggest that you obtain a Google Nexus 6 smartphone to use as a development device. Happily they are not terribly expensive.

PhoneLab Branch Philosophy

Before you go ahead and make changes to PhoneLab platform, we recommend you to at least read this page to get some idea on these two question:

1. Where is my experiment branch based on?
2. How will my changes be merged?

This diagram shows how we manage branches for our platform at high level. You can find more details next.

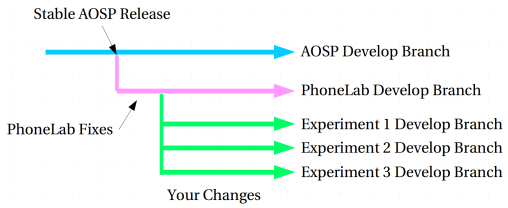


Fig. 1.2: PhoneLab Platform Branching Diagram.

PhoneLab Develop Branch

When there is a major release of Android, we usually wait some time for it to be become mature enough. Then we choose a fairly stable release for our device (Nexus 5) and create our develop branch from there. The branch name is `phonelab/$tag/develop`, where `$tag` is the AOSP release tag. For instance, we used the release `android-4.4.4_r1` for KitKat, `android-5.1.1_r3` for Lollipop, and `cm-13.0` for Mashmallow.

Our develop branch usually only contains device specific changes to make the platform fully functional, including GPS, cellular—parts there are usually broken in original AOSP platform. The develop branch serves as a common ground and does not contain any instrumentations or experiment specific changes.

Experiment Branches

To support parallel development of multiple experiments, we create a branch for each experiment on top of our PhoneLab develop branch. The experiment branch name is in the format of `experiment/$tag/$id/$name`:

1. `$tag` is the base AOSP release tag that our PhoneLab develop branch is based on, for instance, `android-5.1.1_r3` or `cm-13.0`.
2. `$id` is an integer that uniquely identifies your experiment.
3. `$name` is a code name for your experiment, which is determined by you.

By default, your experiment branch is not publicly available: only PhoneLab administrators and yourself have full access to the branch.

Deployment

When we deploy your experiment, we will create a release branch from our PhoneLab develop branch, and merge your experiment branch into that release branch.

Since we may continue development on our develop branch AFTER we create the experiment branch for you, **it is your responsibility to make sure that the merging finishes smoothly without conflicts.** You can ensure this by

trying to merge our develop branch into your experiment to resolve any conflicts beforehand, so that the merging on our part is just a fast-forward.

Here are steps to make sure your changes can be successfully merged.

First, fetch the latest PhoneLab develop branch.

```
$ repo forall -j 8 -pvec git fetch aosp phonelab/$tag/develop:phonelab/$tag/develop
```

Second, make sure your are in your experiment branch.

```
$ repo forall -j 8 -pvec git checkout experiment/$tag/<id>/<name>
```

Finally, merge PhoneLab develop branch into your experiment branch.

```
$ repo forall -j 8 -pvec git merge phonelab/$tag/develop
```

You may need repeat the last step a couple of times to fix possible conflicts.

Warning: NEVER NEVER merge any other branches (e.g., release branches, other experiment branches, logging branches) into your branch. Your experiment branch should only contains your changes!

Warning: If your experiment branch can not be merged into our release branch, it will be excluded from the release.

We developed a tool that will check whether or not your experiment changes meets the above requirement. **Please make sure your pass the check before notifying us your changes are ready.**

https://github.com/blue-systems-group/project.phonelab.platform_checker

Experiment Logistics

Guidelines

Experimenting with the platform image running on several-hundred actual smartphones is risky, so our goal is to make this possible but not necessarily easy. Keeping the following guidelines in mind as you make your experimental changes will help:

- **Don't break stuff.** The fastest way to lose our confidence is to provide us with changes that don't build or cause parts of the our PhoneLab image to fail. There's a road back from this point, but it's uphill. Test your changes thoroughly before submitting them to us.
- **Make useful and novel changes.** Given the dangers associated with this kind of experimentation we are expecting researchers to approach us with exciting and novel ideas that could potentially benefit PhoneLab participants. (In that case, your changes will live on forever as part of the base system!)
- **Be patient.** This isn't a fast process and it's not designed to be. If you have a paper deadline in a week—or even a month—forget it. Your scheduling constraints are your problem—keeping our participants safe is ours.

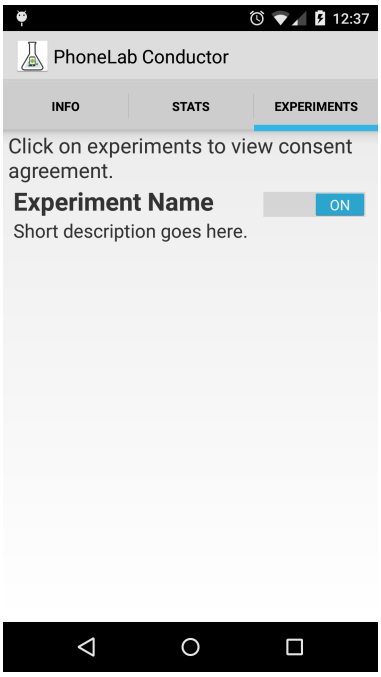
Experiment Information

When your experiment changes have been successfully merged and build, we need these extra information from you about the experiment:

1. Experiment name

2. Short description (a few sentences)
3. Consent URL: a small web page explains what kind of data that your experiment will collect.

These information will be presented to PhoneLab participants as follows:



Participants will have the option to “opt-out” your experiment data collection, which means the data from their devices will not show up in the final dataset we release.

Release Schedule

We will first push the platform changes to a small group of PhoneLab developers, leave it running for at least a week to ensure the changes does not significantly affect user experience. After that, we will push the changes to the entire testbed. So please **expect at least two weeks delay** from when your changes are ready to when they are deployed on the whole testbed.

To avoid issues with cross-modification compatibility or interference, we plan to generally only distribute one experimental modification on PhoneLab at any given point in time. The PhoneLab team will work with you to schedule your experiment to ensure that it receives enough time on the testbed to generate useful data. If your changes are unintrusive and useful, they may stick around, but we provide no guarantees of this. We do guaranteed that if we receive complaints from participants or notice stability issues your changes will be reverted immediately.

Logging Infrastructure

This page answers these questions:

1. How does PhoneLab collect data?
2. How do I add new logging to PhoneLab platform?
3. What has already been logged?

Log Data Collection

We rely on Android Logcat system for data collection. Basically, you use the Log class to write log messages, and we have a small daemon app, PhoneLab Conductor, running on the device, which will collect the log messages and upload them to our backend server. More specifically, **we are collecting the ‘‘main‘‘ logcat buffer using ‘‘threadtime‘‘ format.** Please see here for explanation about the logcat buffers and formats.

We made some improvements to the `Logcat` system:

1. You can log up to 64K characters in one line. Android's default limit is 4K.
2. We enlarge the kernel's buffer for logcat messages from 256KB to 16MB, to tolerate more aggressive logging you may come up with, and also avoid losing log messages due to buffer overflow.
3. We developed a new daemon service, called `kmsgd`, which pipes all kernel log messages (logged by `printk`) to the logcat buffer, which then will be collected and uploaded by PhoneLab Conductor.
4. We add `year` information to the logcat's `threadtime` format.

Guidelines for Logging

When adding instrumentation to the PhoneLab platform there are really only a few rules and guidelines that we ask experimenters to follow:

1. **Follow the logging conventions below.** This helps us keep things organized and avoid duplicated effort. You can also check out the existing tags for examples providing more guidance on our logging conventions.
2. **Record something new and interesting.** Check our existing tags before you get started. If it's already being logged, then someone did part of your job for you! Proceed directly to data request and analysis. The only exception to this rule is cases where the platform is already logging information but poorly-formatted—in this case duplicated logging using JSON for easy deserialization may be worth it.

3. **Remember: Android is more than services.** The PhoneLab Android platform includes many interesting libraries as well as pre-packaged apps such as an alarm clock, calculator, and the Chrome web browser. These may be very interesting places to add instrumentation.
4. **Log intelligently, particularly when adding instrumentation to hot code paths.** Android can support fairly high-volume logging, but please be intelligent when adding instrumentation that could potentially produce a great deal of output as this also slows down post-processing. Do you really need to log every screen redraw separately? Maybe recording the number of redraws per second and logging that once per second is sufficient. If you have questions, contact the PhoneLab team.
5. **Log using JSON.** JSON deserialization libraries exist for almost every useful post-processing language. Writing tools based on regular-expressions is tedious and terrible.

Log Format Conventions

We describe the tag, message, and commenting conventions we ask experimenters to follow. Our goal is not to annoy you—it is to make log messages easy to process and categorize, and to ensure that we avoid duplication of effort between different research teams.

The tags should consist of three parts: (1) category, (2) subcategory, (3) and a short organizational identifier, connected using a dash ('-'): `Category-Subcategory-Organization`. We use `PhoneLab` as the organizational code for the log messages added and maintained by the core PhoneLab team. Here are a few examples:

- `Network-Wifi-PhoneLab`: collects network information about the Wifi interface, added by PhoneLab.
- `Power-Screen-PhoneLab`: collects power information related to screen usage, added by PhoneLab.

Use common sense to determine the category and sub-category field, but don't get too bogged down here—this is our best attempt at a taxonomy, and it's far from perfect. (For example: the `Power-Screen-PhoneLab` tag above could also be under a `Usage` category, since the screen power state indicates whether foreground or background activity is occurring.) You can check current category and sub-category information of existing tags to determine how your new instrumentation fits in. If none of the categories or sub-categories meets your needs, feel free to propose your own.

The content of the log message should be a JSON string, making your results easy to deserialize by a variety of downstream tools. We provide some helper classes to assist in this process.

PhoneLab Log Helper

To simplify logging JSON strings, we provide a set of of helper classes and interfaces in `frameworks/base/core/java/edu/buffalo/cse/phonelab/json`. One interface and two helper classes are defined:

- `JSONable`: Any class that implements this interface must provide a `toJSONObject` method, which returns a `StrictJSONObject` representing this object.
- `StrictJSONObject` and `StrictJSONArray`: They are similar to `org.json.{JSONObject,JSONArray}` but only accepts `JSONable` objects as values. Please see comments at the head of `StrictJSONObject.java` for details.

For example, this is a code snippet that we added to log Wifi scan results:

```
import edu.buffalo.cse.phonelab.json.StrictJSONObject;

// somewhere in the code

  (new StrictJSONObject(PHONELAB_TAG))
    .put(StrictJSONObject.KEY_ACTION, WifiManager.SCAN_RESULTS_AVAILABLE_ACTION)
```

```
    .put("Results", mScanResults)
    .log();
```

Note that:

1. When appropriate, each `StrictJSONObject` should have a `StrictJSONObject.KEY_ACTION` key to differentiate different types of output logged under the same tag. For example, we could use the common `Network-Wifi-PhoneLab` tag to log multiple Wifi-related events associated with different actions—scan results, Wifi connections and disconnections, etc—using a different action attribute for each. **In fact, an exception will be thrown if the StrictJSONObject does not have an action key when its ‘‘log‘‘ method is called.**
2. You can only put `JSONable` objects into `StrictJSONObject`, which means if the object you want to log does not implements `JSONable`, you'll have to implement by yourself. It's not that difficult as it sounds like, please see `ScanResult.java` for an example.
3. When its `log` method is called, the `StrictJSONObject` will add a `timestamp` field if it does not exist already. This is to help you determine ordering between log messages, or want the exact timestamp when some event happened.

Document Your Logs

To help us keep an record of what have been logged, we require you comment you logs in a specific way so that we can automate the process of traversing the whole source tree building a complete log taxonomy. This is an example comment for the Wifi scan results tag described above:

```
/**
 * PhoneLab
 *
 * {
 * "Category": "Network",
 * "SubCategory": "Wifi",
 * "Tag": "Network-Wifi-PhoneLab",
 * "Action": "android.net.wifi.SCAN_RESULTS",
 * "Description": "Wifi scan results."
 * }
 */
(new StrictJSONObject(PHONELAB_TAG))
  .put("Action", WifiManager.SCAN_RESULTS_AVAILABLE_ACTION)
  .put("Results", mScanResults)
  .log();
```

Note that:

1. The first two lines must match the example above exactly. They are the anchor point for our tag processing script.
2. The main body of the comment should be a JSON string, with the five keys in the example. Any extra keys will be ignored. Any $\star$ symbol inside the JSON string will also be ignored.

Log In C/C++ World

We recommend you to use the convenient `StrictJSONObject` whenever you are instrumenting Java sources. If you are working in lower level C or C++ files here are some instructions that you may find helpful.

The header file you need to include for Android logcat support is located in `system/core/include/log/log.h`. The main function you will use is `__android_log_buf_write`. It takes 4 arguments:

1. `bufID`: Android logcat buffer id. Must be one of `LOG_ID_{SYSTEM, MAIN, RADIO, EVENTS}` constants.
2. `prio`: Log line priority. Must be one of `ANDROID_LOG_{VERBOSE, DEBUG, INFO, WARN, ERROR, FATAL}` constants.
3. `tag`: Tag name. Please use our tag name convention described above.
4. `msg`: The message body you want to log. Please use a JSON string.

You can also use the more friendly `__android_log_buf_print` to get `printf` style string formatting.

Logging in the Kernel

On PhoneLab builds, there is a daemon (`kmsgd`) that collects everything logged from the kernel using `printk`, under the tag `KernelPrintk`. To distinguish your logs from other kernel logs, we have provided functionality that can be accessed by adding `#include <linux/phonelab.h>` to the kernel files you're modifying. Using these functions, `kmsgd` will ensure your kernel logs are assigned the appropriate tags.

The kernel logging functions are equivalent to using Android's `Log.*` functions and the logs will appear in both `/proc/kmsg` and `Logcat`. The following table shows the available logging functions and their Android counterparts.

Kernel Logging Function	Android Logging Function
`alog_v(char *tag, const char *fmt, ...)`	`Log.v(...)`
`alog_d(char *tag, const char *fmt, ...)`	`Log.d(...)`
`alog_i(char *tag, const char *fmt, ...)`	`Log.i(...)`
`alog_w(char *tag, const char *fmt, ...)`	`Log.w(...)`
`alog_e(char *tag, const char *fmt, ...)`	`Log.e(...)`

Function Argument Notes:

- `tag` should use the same *Log Format Conventions*
- The functions are `printk` style and can include a variable number of arguments
  - `fmt` is the format string, which should also be a JSON string
  - The current length limit of the output JSON string, after format substitution, is 4096 characters
  - You do not need to add a `'\n'` to `fmt`

The kernel time will be appended to the JSON string with the key `KTime`. You may want to include `SystemClock.uptimeMillis()` in your Android logs in order to more tightly integrate the logs.

Uploading Raw Files

Sometimes it may be convenient to be able to update raw data files, such as packet traces. Therefore, we also provide a `FileUploderService` in addition to the text-log collection mechanism. You can see an example on how to use this service at this project.

Existing Instrumentation

We expect our PhoneLab platform to include an increasing amount of instrumentation added both by PhoneLab developers and by external research teams. If our build already contains instrumentation recording what you're interested in, you can proceed directly to requesting data.

Logging Branches

Instrumentations are stagged in their respective branches by category, such as network, location, or power. Here is a list of current logging branches:

1. `logging/android-5.1.1_r3/1/network`
2. `logging/android-5.1.1_r3/2/power`
3. `logging/android-5.1.1_r3/3/location`
4. `logging/android-5.1.1_r3/4/packagemanager`

Add Your Instrumentation

To add instrumentations to these branch, for example, `logging/android-5.1.1_r3/1/network`, please follow these steps:

First, if you have not cloned the repository yet:

```
$ cd <WORKING_DIRECTORY>
$ repo init -u ssh://<USERNAME>@platform.phone-lab.org:8080/platform/manifest -b logging/android-5.1
$ repo sync
```

The `<USERNAME>` is your user name on our Gerrit server.

Next, figure out the repository which you want to add instrumentation, say `frameworks/base`, create a working branch:

```
$ cd frameworks/base
$ git checkout -b my_instrumentation
```

Then you add the instrumentation, commit and upload your changes for review:

```
$ git commit -a -S
$ git push aosp refs/for/logging/android-5.1.1_r3/1/network
```

Note: Note the remote branch name when you push: it is a special Gerrit pseudo branch for changes to be reviewed.

Here is a list of existing instrumentations on our platform.

Summary

PhoneLab's instrumented Android platform currently contains:

- 11 tags, 20 actions,
- ... in 9 categories,
- ... added by 2 institutions.

Catetory: Activity

Tag: `Activity-LifeCycle-QoE`

1. **Action**: `onStart, onPause, onResume`
   Project: frameworks/base
   File: core/java/android/app/Activity.java:1146
   Description: Activity lifecycle events

Catetory: Location

Tag: `Location-Misc-PhoneLab`

1. **Action**: `android.location.LOCATION_CHANGED`
   Project: frameworks/base
   File: services/core/java/com/android/server/LocationManagerService.java:2252
   Description: Location update.

Catetory: Network

Tag: `Network-Telephony-PhoneLab`

1. **Action**: `android.intent.action.ANY_DATA_STATE`
   Project: frameworks/base
   File: services/core/java/com/android/server/TelephonyRegistry.java:1592
   Description: Cellular data connectivity changed.
2. **Action**: `android.intent.action.DATA_CONNECTION_FAILED`
   Project: frameworks/base
   File: services/core/java/com/android/server/TelephonyRegistry.java:1649
   Description: Cellular data connection failed.
3. **Action**: `android.intent.action.PHONE_STATE`
   Project: frameworks/base
   File: services/core/java/com/android/server/TelephonyRegistry.java:1547
   Description: Phone calling state changed (incoming call).
4. **Action**: `android.intent.action.SERVICE_STATE`
   Project: frameworks/base
   File: services/core/java/com/android/server/TelephonyRegistry.java:1466
   Description: Cellular service state changed.
5. **Action**: `android.intent.action.SIG_STR`
   Project: frameworks/base
   File: services/core/java/com/android/server/TelephonyRegistry.java:1501
   Description: Cellular signal strength changed.
6. **Action**: `android.telephony.CALL_FORWARDING_CHANGED`
   Project: frameworks/base
   File: services/core/java/com/android/server/TelephonyRegistry.java:1027

Description: Call forwarding status changed.

7. **Action**: `android.telephony.CELL_LOCATION_CHANGED`
   Project: frameworks/base
   File: services/core/java/com/android/server/TelephonyRegistry.java:1257
   Description: Cell tower location changed.
8. **Action**: `android.telephony.DATA_ACTIVITY_CHANGED`
   Project: frameworks/base
   File: services/core/java/com/android/server/TelephonyRegistry.java:1070
   Description: Cellular data activity.
9. **Action**: `android.telephony.MESSAGE_WAITING_CHANGED`
   Project: frameworks/base
   File: services/core/java/com/android/server/TelephonyRegistry.java:979
   Description: Message waiting status changed.

Catetory: PackageManager

Tag: `PackageManager-Misc-PhoneLab`

1. **Action**: `android.intent.action.PACKAGE_{ADDED, CHANGED, REMOVED}`
   Project: frameworks/base
   File: services/core/java/com/android/server/pm/PackageManagerService.java:10154
   Description: Package installed/uninstalled/updated.

Catetory: Power

Tag: `Power-Battery-PhoneLab`

1. **Action**: `android.intent.action.BATTERY_CHANGED`
   Project: frameworks/base
   File: services/core/java/com/android/server/BatteryService.java:661
   Description: Battery status changed.

Tag: `Power-Screen-PhoneLab`

1. **Action**: `android.intent.action.SCREEN_OFF`
   Project: frameworks/base
   File: services/core/java/com/android/server/power/Notifier.java:634
   Description: Screen turned off.
2. **Action**: `android.intent.action.SCREEN_ON`
   Project: frameworks/base
   File: services/core/java/com/android/server/power/Notifier.java:596
   Description: Screen turned on.

Catetory: Spinner

Tag: `Spinner-State-QoE`

1. **Action**: `ProgressBarEvent`
   Project: frameworks/base
   File: core/java/android/widget/ProgressBar.java:1505
   Description: Start and end of indeterminate progressbars

Catetory: Usage

Tag: `KeyEvent-UserAction-QoE`

1. **Action**: `HardwareTouchEvent`
   Project: frameworks/base
   File: core/java/android/view/KeyEvent.java:1594
   Description: User pressed a key

Catetory: View

Tag: `View-UserAction-QoE`

1. **Action**: `TouchEvent`
   Project: frameworks/base
   File: core/java/android/view/View.java:10294
   Description: User touched item

Catetory: WebView

Tag: `WebView-Update-QoE`

1. **Action**: `WebViewUpdateEvent`
   Project: frameworks/base
   File: core/java/android/webkit/WebViewClient.java:38
   Description: Webview loading progress
2. **Action**: `WebViewUpdateEvent`
   Project: frameworks/base
   File: core/java/android/webkit/WebChromeClient.java:39
   Description: Webview loading progress

Tag: `WebView-UserActino-QoE`

1. **Action**: `WebViewTouchEvent`
   Project: frameworks/base
   File: core/java/android/webkit/WebView.java:304
   Description: User touched item in webview

Last updated 2016-12-02

Data Release

Sample PhoneLab Dataset

We put up a sample dataset that contains one month (March 2015) of data that were collected from 11 PhoneLab devices. You can download the tarball here. The tags in this sample dataset are listed in the existing tags page.

We hope this sample dataset can give you an idea of what the data looks like and help you determine whether or not you want to proceed to the IRB process to request the full dataset. You will also have a chance to start developing your data analysis scripts immediately in parallel to the IRB process.

Request More

Once you've download and built our PhoneLab Android sources, added your experimental instrumentation or platform modifications , and your changes have been deployed, the final step is to request data generated by your experiment.

We will make a web form available to simply this process. However, here are the things you need to provide in order for us to create an archive for you:

1. **Institutional Review Board (IRB) Letter**. You must provide documentation that your request for data and plan to process and publish that data have been reviewed and approved for human subjects safety by your institutions IRB or equivalent body. Your IRB documentation must match the rest of the parameters of your request listed below.
2. **Consent procedure**. If your experiment requires affirmative consent from participants PhoneLab will assist you in performing that request to our participants. No data will be provided from participants that fail to complete the required consent process.
3. **Date Range**. Including start day and end day, inclusive. Days are the granularity by which we organize our archives so finer divisions (such as hours) are not possible.
4. **Tag List**. A list of all tags to return data from.

We are aware that IRB standards and procedures vary considerably between institutions, and it is possible that getting your experiment approved may take some time and effort on your part. However, we cannot assist you with this process nor will any exceptions be made to our IRB approval requirement.

Once you have an archive this documentation will help you understand the archive structure and file format.

Distributing Surveys

Here are steps to distribute a survey of your experiments to PhoneLab participants.

1. Compose your questionnaire using a survey provider. Note that **the provider need to support embedded user ID in the URL**. SurveyGizmo is know to support this feature.
2. Send us a **survey URL** and a **email template** that we will forward to each individual participants.
3. We will substitute the user ID placeholder in the survey URL with the device's hashed ID, and send an email to each participant with this URL and the email template you send us earlier.

Note: To incentivize participation, we suggest you also provide prizes for the survey, such as Amazon Gift card. After the survey, you can draw winners from those who completed the survey, and we will send the prize to corresponding participant.

PhoneLab Data Format

Once you have received the data you requested the next step is to process it. PhoneLab provides some of our own tools as is to help with this process and we encourage experiments to reuse and contribute to them. However, if you have your own tools the following documentation on the structure of our data archives and files will be helpful.

Archive Format

The resulting `TAR` archive we will provide to IRB-approved experimenters consists of one file per device per tag per day structured as follows: `device_identifier/tag/year/month/day.out.gz`, so as an example `b3793ae1229920c02b564adbc200780168cd42ed/Location-Misc-PhoneLab/2014/11/19.out.gz`. When generating these these files we have attempted to ensure the following:

- **All log messages are captured.** Our data collection tools make every effort to recover all generated log messages: including configuring large Linux log buffers in our platform image and caching up to several days worth of log messages on each device between uploads. However, these measures are obviously not foolproof and experimenters are encouraged to implement their own reliability mechanisms as needed to detect missing log tags. Also please see the note below on one important and well-known source of missing logs: early boot.
- **Each file is sorted by time.** However, this is complicated by the fact that in certain cases `logcat` can generate multiple log lines with identical timestamps—particularly if the logged data contains newlines. In the case of identical timestamps we defer to the order in which the lines were originally logged by `logcat`, and when processing identical timestamps split across multiple files, the order in which the files were uploaded to the PhoneLab backend.
- **Duplicate log lines are omitted.** Our efforts to recover all log messages sometimes lead to duplicate logs being uploaded or log files being processed twice, but we attempt to remove duplicate messages during post-processing. However, this is complicated by the lack of timestamp uniqueness described above, which we work around in our logging helper classes using a unique ID embedded in the JSON message. However, because deduplication is done during log processing only using the timestamp fields, duplicate messages may exist in the archive.

Missing Early Boot Logs

One well-known source of missing log messages are from messages generated early during Android boot. The problem arises because at this point the device does not yet have a network-provided date and time, and so log messages are timestamped as being generated in 1970—at the beginning of the Unix epoch. It would probably be possible to fix this problem by retroactively correcting early boot log message timestamps once a network-provided time is available, but have yet to implement this fix. At present, these timestamps will be (correctly) sorted into a 1970 tag file but (incorrectly) intermingled with many other log tags also generated during other boot cycles.

If you are running an experiment that requires early boot logs, please feel free to contact the PhoneLab team and we will see if we can come up with a better solution to the problem together. For now these logs will simply be omitted from all archives.

File Format

Each line in the file begins with the following standard fields. If you have worked with `logcat` before, this will look familiar to you, as many of the fields are taken directly from the `logcat` output. We describe each of the standard fields in more detail below, using several examples based on actual log messages uploaded by PhoneLab Director Geoffrey Challen's device.

Device Identifier	UNIX Timestamp	Ordering	Date and Time	Process ID	Thread ID	Log Level	Tag
7699f273	1416261509170	1416261509170.140	2014-11-17 21:58:29.970997	769	1026	I	Power-Battery-PhoneLab
7699f273	1416261509170	1416261509170.141	2014-11-17 21:58:29.970997	879	1143	D	Location-Misc-PhoneLab

1. Device Identifier: A unique identifier for each device.
2. UNIX Timestamp: Milliseconds since 1970. Note even at this resolution this timestamp is not guaranteed to be unique across all log messages, creating the need for the next field.
3. Ordering: This field takes the form `milliseconds_since_1970.order_in_upload_file`. For log messages that do not share a timestamp with any other line, it will be `milliseconds_since_1970.0`. In other cases it will be set as shown in the two examples above. Note that this is only sufficient to provide an ordering for identically-timestamped messages in the same file; cross-file ordering is still not handled properly by our tool chain. Also note that this example is contrived as identical timestamps occur most often due to (1) multiple `logcat` messages on neighboring lines of the same app or (2) `logcat` messages that contain newlines.
4. Date and Time: Human-readable date and timestamp using the device's locale. In this case the timestamps are in Eastern Standard Time (EST).
5. Process and Thread IDs: Fairly self-explanatory.
6. Log Level: Android uses verbose (V), debug (D), info (I), warning (W) and error (E) log levels on a per-message basis. This page has more details.
7. Tag: Log messages are either generated by the instrumentation we have added or by existing logging included in the Android platform by default or left enabled by many apps after deployment—despite Android's suggestions to the contrary.

These fields are following by a log message as a single string, which can be up to 64k characters long—but hopefully nowhere close to that limit! Obviously the format of the log string varies based on what is being recorded, but here are a few examples. First, a JSON-formatted log string generated by our tools under the `Power-Battery-PhoneLab` tag, with internal fields that are self-explanatory:

```
{
    "Action":"android.intent.action.BATTERY_CHANGED",
    "LogFormat":"1.0",
    "BatteryProperties":{
        "Status":"Charging",
        "Present":true,
        "Voltage":4342,
        "Temperature":255,
        "CurrentNow":-794372,
        "Health":"Good",
        "Level":94,
        "PlugType":"AC",
        "ChargeCounter":-2147483648,
        "Technology":"Li-ion"
    },
```

```
    "Scale":100
}
```

And second, an example of something not formatted in JSON—in this case, garbage collection output generated under the `dalvikvm` tag:

```
GC_FOR_ALLOC freed 259K, 6% free 18632K/19680K, paused 16ms, total
16m
```

PhoneLab Cruncher

Todo

revise this section about cruncher.

The PhoneLab `cruncher` is our own early attempt to produce a reasonably-efficient and kind-of user-friendly set of log post-processing tools. You are welcome to download, use, and modify it to suit your needs—just don't expect us to support it. It should already support many of the log tags we have added to the PhoneLab platform, particularly ones we have used for our own experimental purposes.

The `cruncher` (ab)uses the Django object-relational mapper (ORM) to ease the process of manipulating a database in Python. Given that (1) importing logs from the files into the database and (2) processing the logs further to produce useful output are both potentially time-consuming, the `cruncher` splits log import and processing into three phases with different parallelization constraints, each of which can be repeated as needed during post-processing tool development:

1. **Log import:** the process of importing logs from the flat files into the database is parallelized by log file, meaning that logs can be processed in any order in any queue. As a result, no relationships between log tags can be established or used during import. Instead, each log line should generate one (or many) database objects. Django's transaction and bulk loading support are used to make this relatively quick.
2. **Per-device processing:** the second stage is parallelized by device and provides the opportunity to combine information from multiple log messages. For example, separate messages logged during file `open()` and `close()` along with information about intervening `read()` and `write()` operations could be combined to create a single file session object. However, at this stage no cross-device relationships or statistics can be computed. The `cruncher` provides several different optimized iterators allowing code to loop over one or more of the objects created during the import stage—but again, strictly on a per-device basis.
3. **Final processing:** once all per-device processing has completed `cruncher` code has access to all models from all devices and can compute overall statistics or generate graphs integrating data from the entire experiment.

Currently the `cruncher` is capable of making efficient use of multiple cores to maximize IO throughput when importing and processing logs, but not yet of using multiple machines to further parallelize the process. We are actively working on this feature. If you would like to help, we would welcome the assistance.

PhoneLab Policies

PhoneLab is an open-access smartphone platform testbed operated by the PhoneLab team (we) at the Computer Science and Engineering Department at the University at Buffalo, The State University of New York (UB). It consists of smartphones, participants that use the smartphones, and control software. It is designed to be open-access, meaning that researchers inside and outside of UB will conduct research studies using PhoneLab. It provides realism by having participants use their experimental smartphone as their primary device allowing research experiments to be performed under realistic conditions and with real user inputs. This document contains terms and conditions that each and every experimenter should be aware of and agree to before using PhoneLab.

Access and Administration

The use of the testbed is allowed for most academic purposes. We will reserve the right to grant or deny PhoneLab access depending on the nature of the experiment and the demand of the testbed. The current policy only allows the use of the testbed for research, not for education or development.

Data Release

PhoneLab will only release data to researchers whose requests have been reviewed and approved or marked exempt by their institution's Institutional Review Board (IRB) or equivalent human subjects protection body. PhoneLab will assist researchers whose protocols require affirmative consent from PhoneLab participants.

Participation

PhoneLab makes no guarantees about the size of our testbed or how many participants will choose to provide data at your request. Participants may opt in and out of experiments on a per-experiment basis.

Acceptable Use

We require that researchers act within the bounds of the experiment they have had reviewed by their IRB. We reserve the right to define abuse of PhoneLab as broadly as possible in the interest of protecting our participants. Misrepresentation of the purpose for using the testbed or misuse of PhoneLab data will result in experimenters losing access to PhoneLab and having their academic or research institution notified of their behavior.

Policy Changes

We also reserve the right to change our policies at any point of time in the future.

Disclaimer

PhoneLab is provided as-is without any warranty. We disclaim any and all liability of an experiment's results, its errors, and any of its side-effects, broadly defined. We also disclaim any errors or problems that arise out of security and reliability issues of our software.

Frequently Asked Questions

What is the device ID in the log lines?

The device ID is a string of length 40. The value is a hex digest of hashed `TelephonyManager.getDeviceId()` (link). A sample code snippet to get the device ID in Java is as follows:

```
TelephonyManager telephonyManager = (TelephonyManager) context.getSystemService(Context.TELEPHONY_SE
MessageDigest digester = MessageDigest.getInstance("SHA-1");
byte[] digest = digester.digest(telephonyManager.getDeviceId().getBytes());
deviceID = (new BigInteger(1, digest)).toString(16);
```

What happed if the participant opt out my experiment?

Note that your experimental changes will be pushed to participants' devices and will be running during the experiment period **no matter** whether the user opt out your experiment or not. That choice (opt-out) will be recorded by us and only used to determine whether we should release the data collected from that particular participant's device to you (experimenter).

Do I need to log a event if it's already logged in other branch?

It depends. If the event is logged in one of the `logging/android-5.1.1_r3/...` branches, then NO NEED to log it again, since these branches will almost always be included in the release branch. In fact, we strongly advise you NOT to repeat the effort, as it will potentially cause merge conflicts.

However, if the event is logged in another experiment branch (e.g., `experiment/android-5.1.1_r3/...`), then we suggest you log the information again in your branch, since the other experiment may end earlier than yours.

Shall I merge other logging or experiment branches into my branch?

You should never do this. The only branch you need to keep up to date is the PhoneLab develop branch (`phonelab/android-5.1.1_r3/develop`).

Experiments

PhoneLab is a public smartphone testbed. We solicit researchers with exciting new ideas to experiment on PhoneLab that are made possible with PhoneLab's ability to modifying the AOSP platform. Over the years, PhoneLab has facilitated the following smartphone platform experiments.

Ongoing

Completed

1. DefDroid

9/21/2015 - 11/3/2015

The goal of DefDroid is to make the mobile OS more defensive to curb the naughty apps that drain your battery or over-consume your mobile data, storage, etc. We design DefDroid so that it makes your mobile phone more sustainable without breaking the main functionalities of the apps.

Contact:

Ryan (Peng) Huang (Advisor: Prof. Yuanyuan Zhou)
UCSD

2. Lock Screen

10/22/2015 - 6/3/2016

This experiment looks at how users interact with their lock screens. We collect log information on whether a code-based lock is enabled, how much time is spent before unlocking the device, how long users take to enter the code and

how many failed attempts occur. This information will help researchers to design lock screens with better security while maintaining or improving upon existing usage patterns.

Contact:

Marian Harbach (Advisor: Serge Egelman)
ICSI @ UC Berkeley

3. LTE Handover Analysis

10/28/2015 - 6/3/2016

This experiment aims to study the decision policy and performance impact of handovers including WiFi-Cellular handover, IRAT (Inter radio access technology) handover, and intra-LTE handover.

Contact

Shichang Shawn Xu (Advisor: Prof. Z. Morley Mao)
University of Michigan, Ann Arbor

4. Runtime Permission

11/24/2015 - 3/16/2016

This is a study on privacy preferences of mobile users when it comes to sensitive data requests originating from third party applications. To that end, we want to track sensitive data requests and ask users whether they want to block such requests as it happens. However we hope to prompt the question at most once per day per user when such a request occurs. We are also hoping to log surrounding contextual data when such a question is prompted to the user.

Contact:

Primal Wijesekera (Advisor: Prof. Konstanin Beznosov)
UC Berkeley & University of British Columbia

5. Maybe

11/13/2015 - 11/24/2015

One of the reasons programming mobile systems is so hard is the uncertainty created by the wide variety of environments a typical app encounters at runtime. In many cases only post-deployment user testing can determine the right algorithm to use, the rate at which something should happen, or when an app should attempt to conserve energy. Programmers should not be forced to make these choices at development time. But today's programming languages leave no way for programmers to express and structure their uncertainty about runtime conditions, forcing them to adopt ineffective, fragile, and untested ad-hoc approaches to runtime adaptation. We introduce a new approach based on structured uncertainty through a new language construct: the maybe statement.

Contact:

Yihong Chen (Advisor: Geoffrey Challen)
University at Buffalo

6. File System Analysis

11/3/2015 - 11/13/2015

Centralized cloud storage services such as Dropbox have revolutionized the way that users share files and access data across their growing number of devices. But today's cloud storage options have serious limitations affecting mobile

battery-powered smartphones. Many central cloud storage providers require each client to have enough storage for an entire replica, which may not be feasible on smartphones with an order-of-magnitude less storage than laptops and desktops. Centralized cloud storage does not scale as users add more storage and misses the opportunity to harness free space users already have. And centralized cloud storage provides poor support for mobile devices, both failing to leverage natural mobility patterns when distributing data and potentially causing costly mobile data traffic.

Contact:

Carl Nuessle (Advisor: Geoffrey Challen)
University at Buffalo

7. Quality of Experience

11/3/2015 - 11/16/2015

Of all the resources that smartphones manage, human attention is the most precious. While processor speed and core count, memory and storage capacity, and network bandwidth have steadily and sometimes rapidly increased, the number of hours in the day has not. And as users spend an increasing amount of time with their personal computing devices, it is more important than ever that these devices ensure that their time is used effectively. We refer to this as quality of experience (QoE).

Contact:

Scott Haseley (Advisor: Geoffrey Challen)
University at Buffalo

8. Jouler

3/7/2016 - 3/16/2016

Despite the fact that current smartphone platforms already incorporate energy measurement tools and multiple energy control mechanisms, smartphone battery lifetimes continue to frustrate users. This is because measurements and mechanisms are of limited utility without policies that utilize them to achieve different energy management goals, such as meeting a lifetime target or providing good performance to a user's favorite apps. To address this problem we are developing Jouler, a policy framework enabling effective and flexible smartphone energy management.

Contact:

Anudipa Maiti (Advisor: Geoffrey Challen)
University at Buffalo

9. Bluetooth Low Energy

11/03/2015 - 8/31/2016

We collect information that nearby BLE powered devices publicly broadcast. This enables us to study the privacy threats they pose. Please make sure you keep the Bluetooth radio turned on for sometime during the day.

Contact:

Kassem Fawaz (Advisor: Prof. Kang G. Shin)
RTCL @ University of Michigan, Ann Arbor

10. GridWatch: Crowdsourcing the Detection of Power Outages and Restorations

03/03/2016 - 8/31/2016

This experiment is gathering information to validate the GridWatch system. GridWatch is a system that attempts to crowd-source the detection of power outages and power restorations. These events are sensed using unmodified smartphones. The key insight is that when a charging phone stops charging, it might have experienced a power outage. When multiple phones that are nearby each other stop charging at the same time, it becomes more likely that an outage occurred. This same logic applies for power restorations, except instead of stopping charging, phones start charging. This experiment will gather your battery state (charging, not charging, percent charged) and your last known GPS location when battery state changes.

Contact:

Noah Klugman (Advisor: Prabal Dutta)
University of Michigan, Ann Arbor

11. Smartphone Storage Analysis

06/13/2016 - 8/31/2016

The purpose of this study is to determine the amount of storage space consumed by modern mobile apps on smartphones and effect of app usage on storage. The results will help developing the new generation of storage for smartphones and identifying minimum amount of storage space today's smartphones must have.

Contact:

Ashish Bijlani (Advisor: Prof. Roy H. Campbell)
UIUC

12. CPU Thermal Management

03/31/2016 - 8/31/2016

This experiment aims to study the thermal characteristics of smartphones. We monitor the temperature of your smartphones and attempt to detect bad choices made by Android that make the phones run hot. Our goal is to use this information to prevent phones from (unnecessarily) overheating and also improve battery life.

Contact:

Guru Prasad Srinivasa and Scott Haseley (Advisor: Geoffrey Challen)
University at Buffalo

13. QoEye

07/04/2016 - 8/31/2016

QoEye collects high-level interactions with app components to help study Quality of Experience (QoE). Our goal is to discover common app usage patterns and to use this data to replay these interactions, eventually determining the contributing factors of QoE for various apps.

Contact:

Scott Haseley (Advisor: Geoffrey Challen)
University at Buffalo

14. TicToc: User Authentication through UI profiling

07/04/2016 - 8/31/2016

This study will record low-level interaction with the phone to study identifiable user-machine interaction abnormalities that are unique to each user. We hypothesize that this profiling low-level interaction will be useful in detecting impersonation attacks.

Contact:

Ahmed M Fawaz (Advisor: Prof. William H. Sanders)
UIUC

15. M2Auth

07/06/2016 - 8/31/2016

This experiment aims to explore the behavioral biometrics-the way that user interact with the smartphone, such as how user touching the screen instead of what user touch. This data will help us to design a Multi-Modal Authentication framework that incorporate different modalities of these biometrics.

Contact:

Ahmed Mahfouz (Advisor: Prof. Tarek Mahmoud)
Minia University, Egypt