Jouler: A Policy Framework Enabling Effective and Flexible Smartphone Energy Management

Anudipa Maiti, Yihong Chen and Geoffrey Challen

University at Buffalo
{anudipam,ychen78,challen}@buffalo.edu

Abstract. Smartphone energy management is a complex challenge. Considerable energy-related variation exists between devices, apps, and users; and while over-allocating energy can strand the user with an empty battery, over-conserving energy can unnecessarily degrade performance. But despite this complexity, current smartphone platforms include "one-size-fits-all" energy management policies that cannot satisfy the diverse needs of all users. To address this problem we present Jouler, a framework enabling effective and flexible smartphone energy management by cleanly separating energy control mechanisms from management policies. Jouler provides both imperative mechanisms that can control all apps, and cooperative mechanisms that allow modified apps to adapt to the user's energy management goals. We have implemented Jouler for Android and used it to provide three new energy management policies to 203 smartphone users. Results from our deployment indicate that users appreciate more flexible smartphone energy management and that Jouler policies can help users achieve their energy management goals.

Key words: Smartphone energy management; Smartphone platforms

1 Introduction

Effective smartphone energy management requires responding to an enormous amount of diversity. Devices have different battery capacities, users have different battery lifetime expectations determined by their charging habits, and apps consume[1] different amounts of energy depending on what they do and how well they are developed. Despite these differences, today's smartphone platforms manage energy using "one-size-fits-all" policies. For some users, the result is battery lifetimes that are too short, and this has remained a top complaint about smartphones [15, 1]. For other users, the result is battery lifetimes that are unnecessarily long and degraded performance due to unneeded energy conservation.

Recent research efforts have succeeded in improving smartphone energy measurement [4, 20], characterization [16] and modeling [5, 12, 24, 6]. They have also provided new energy control hardware [9, 10] and software [17, 21] mechanisms. However, more accurate measurements and more effective mechanisms will not improve smartphone

[1] To avoid confusion between device usage and energy usage, we use *consumption* to denote energy usage and *usage* to denote user-device interaction.

energy management if they are not joined with a range of different *policies* reflecting the differences between devices, users, and apps.

This paper introduces *Jouler*, a system enabling effective and flexible smartphone energy management. Jouler delegates the energy management policy decisions currently embedded in smartphone platforms to unprivileged apps called *energy managers*. Energy managers use Jouler's interface to access energy measurements and energy control mechanisms. Because energy managers encapsulate energy management policies inside normal smartphone apps, they are easy for developers to create and distribute, and for users to find, try, and rate. They can also interact with the user, monitor the environment, and access all other capabilities provided to apps.

To enable flexible policies, Jouler provides energy managers with a variety of information about running apps. Energy models provide overall and per-app energy consumption measurements, broken down by component and between foreground and background operation. Jouler also provides information about how apps use the device, such as the amount of time they spend in the foreground and their usage of the network, output devices, and sensors.

To enable effective policies, Jouler provides energy managers with both energy control carrots (cooperative mechanisms) and sticks (imperative mechanisms). Jouler's cooperative mechanisms enable cooperation with modified apps that can adapt their own energy consumption when needed, making existing energy-aware apps simpler and more effective by allowing them to offload energy management policy decisions to the energy manager. When cooperation fails, energy managers can utilize imperative mechanisms—such as per-app processor frequency throttling—to force unmodified or uncooperative apps to adjust their energy consumption. Imperative mechanisms also help encourage developers to modify their apps to take advantage of Jouler's cooperative mechanisms.

After motivating our approach using results from a detailed energy consumption measurement study, we present Jouler's design and several potential energy managers. We then evaluate an Android implementation of Jouler in two ways. First, we demonstrate the effectiveness of Jouler's imperative and cooperative control mechanisms on a benchmark app. Second, we present the results of deploying Jouler and three energy managers to 203 PHONELAB participants. Our results show that Jouler is effective and that users appreciate more flexible energy management.

2 Motivation

Jouler's design is motivated by the results of two IRB-approved measurement studies performed on the PHONELAB public smartphone platform testbed [13] located at the University at Buffalo. PHONELAB consists of several hundred students, faculty, and staff who carry instrumented Android smartphones. PHONELAB participants are balanced between genders and distributed across ages, and thus are representative of the broader smartphone user population. Our study both (1) logged battery level changes for 105 users for 6 months and (2) modified the Android platform to record more detailed per-app energy consumption statistics for 107 users for 2 months. Because our results largely match a previous measurement study [3], we summarize them only briefly:

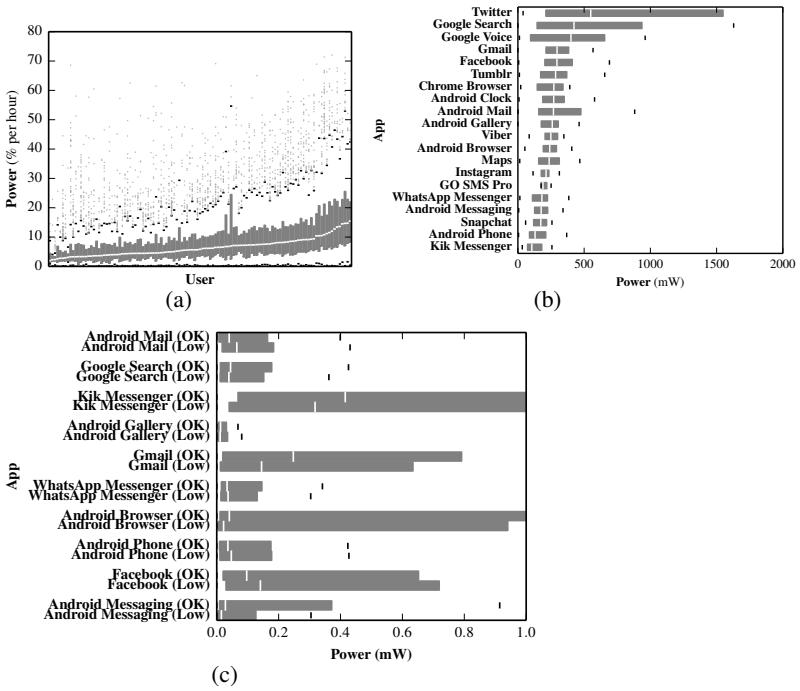


Fig. 1: **Aspects of Energy Consumption Diversity.** In all plots the white line shows the median, shaded bars show upper and lower quartiles, whiskers are positioned at 1.5 times the inner quartile range, and small dots show outliers. Plots show large amounts of interuser (a) and interapp (b) variation, and that apps are not successfully adapting to low battery conditions (c).

– **Interuser variation.** Figure 1a shows per-user distributions of discharging rates (in percent per hour) for all discharging sessions in the six-month trace. A factor of four separates the fastest and slowest users, and a great deal of intrauser variation is visible.
– **Interapp variation.** Figure 1b shows user distributions of per-app energy consumption for the top 20 apps used by PHONELAB users. Because many apps include background services, we compute power by dividing each app's total energy consumption—including both background and foreground—by its foreground time. The data shows a large amount of interapp variation, and, for many apps, a great deal of interuser variation.
– **Apps don't adapt.** To investigate whether apps adapt to low battery levels, we separate measured app energy consumption into low battery ($< 10\%$) and OK battery states and compared these two distributions for the top 10 apps. Because we consider it reasonable for apps to maintain interactive performance even when the battery is low, we only examined background energy consumption for this comparison. Figure 1c shows that in most cases the distributions are very similar, indicating that most apps are not adapting to low battery levels.

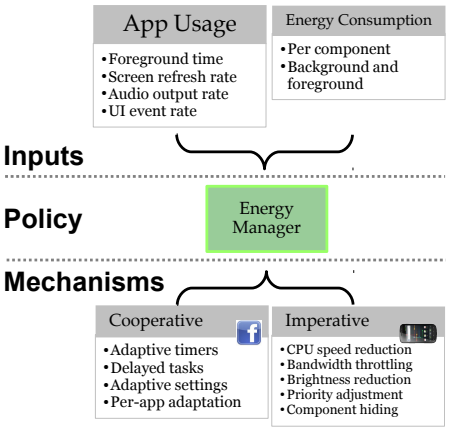


Fig. 2: **The Jouler Energy Management Framework.** Jouler provides energy managers with the information needed to make energy management policy decisions and the mechanisms needed to enforce them.

In summary, analysis of our two datasets confirms the well-known energy consumption differences between users and apps, and motivates the need for more flexible smartphone energy management to respond to this diversity.

3 Design

We continue by describing Jouler's design. Jouler consists of two parts: unprivileged apps called energy managers that implement energy management policies, and a privileged platform service providing an interface to the information and mechanisms used by energy managers to accomplish their goals. We describe each in turn.

3.1 Energy Managers

Enabling flexible energy management requires allowing policies to be easily created and distributed by developers and easily installed, configured, and evaluated by end users. To accomplish this, Jouler utilizes the same solution that has worked so successfully for millions of smartphone apps: app marketplaces like the Google Play Store. Jouler removes energy management policies from within the platform where they cannot be altered and replaces them with category of apps called *energy managers* implementing a variety of different energy management policies. As shown in Figure 2, energy managers use Jouler's interface to access app usage and energy consumption statistics and control per-app and overall device energy consumption.

Because Jouler energy managers are just normal smartphone apps, we have similar expectations for their development and use. We expect a small group of app developers to develop a variety of energy managers to be used by millions of smartphone users. We expect both good, user-friendly, effective as well as complicated, ineffective, malicious energy managers to co-exist. We also expect users to try different energy managers before deciding on one or more making few energy managers more popular than others.

Energy managers are only distinguished from other smartphone apps in two ways. First, they must request and be granted permission to use Jouler's interface. During installation, energy managers request permission to access these features using the platform's standard permission dialog. Second, to prevent multiple energy managers from interfering with each other, Jouler enforces that only one energy manager can be active at any point in time—even if the user has installed several.

To continue, we provide two vignettes presenting energy managers first from the perspective of an energy manager developer and second from that of an end user.

Energy manager developer experience. Alice is an experienced app developer. From personal experience she noticed that while traveling a user is most likely to decrease smartphone usage due to limited charging opportunities to avoid running out of energy too quickly. So she developed a Jouler energy manager that prompts an user to enter her travel plans when it detects her arriving at an airport. It uses her predicted arrival time to determine an appropriate lifetime target, while also prioritizing energy consumption by travel-related apps such as navigational aids. Once Alice is satisfied with her new energy manager, she publishes it to the Google Play Store for other travelers to try.

End user experience Dave and Bob are coworkers who travel together frequently. Dave is a heavy smartphone user and frequent charger and normally uses an energy manager that adapts to his charging habits to provide high performance. Bob, on the other hand, is a light user and forgetful charger and normally uses an energy manager that meters out energy to meet his target lifetime and aggressively reminds him to charge when his battery is low.

Both users, however, have been frustrated by their smartphones' energy consumption when traveling. Searching on the Google Play Store, Bob locates Alice's energy manager which has become popular with travelers. On their next trip, he tries it and finds it effective enough to recommend to Dave, who begins to use it regularly as well. While traveling they enable Alice's energy manager, and when they return home they again enable their normal energy managers.

3.2 Energy Manager Inputs

To enable a variety of effective energy management policies, Jouler provides energy managers with as much information about app usage and energy consumption as possible. To measure energy consumption, Jouler tracks total system and per-app energy consumption, breakdowns of energy consumption between device components (processor, network interfaces, screen, GPS), and breakdowns of energy consumption between screen foreground, audio foreground, and background sessions. While some of this information can be obtained by Android apps through Java introspection, this approach is brittle and not officially supported. Jouler's interface standardizes access to detailed energy consumption information.

To measure interaction, Jouler tracks the number of and length of each app foreground session; rates of click, type, and swipe interactions; screen redraw and audio sampling rates; and notification delivery and click times. This collection of information is sufficient to support the variety of energy managers described later, but Jouler may eventually provide more information if it proves useful to promising new energy management approaches.

3.3 Cooperative Mechanisms

To enable effective energy management policies, Jouler provides energy managers with two types of mechanisms: cooperative mechanisms that rely on collaboration with apps, and imperative mechanisms that do not. Cooperative mechanisms allow apps to guide the process of aligning their own energy consumption with the energy manager's and user's goals. Jouler's collaborative mechanisms combine a simple set of signals with a library of useful energy management primitives based on common app design patterns.

However, imperative mechanisms can always be used to control the energy consumption of apps that either have not been modified to use Jouler or are not cooperating effectively. As a result, no changes to existing apps are required to use Jouler. In addition, because apps have no control over the imperative mechanisms applied to them by the energy manager, imperative mechanisms also serve to incentivize developers to modify their apps to use Jouler's cooperative mechanisms.

Cooperative signals Jouler's cooperative mechanisms are driven by three simple signals that energy managers can send to apps:

- `Reduce` indicates the app must reduce its energy consumption. If it does not, the energy manager may apply an imperative mechanism. This signal is also sent whenever an imperative mechanism is applied.
- `OK` indicates that the app's energy consumption is acceptable to the energy manager.
- `Increase` indicates that the app can increase its energy consumption. This signal can be sent when an imperative mechanism is removed or the device begins charging.

So a cooperative app should immediately reduce its energy consumption on receiving single or repeated *Reduce* signal. Unmodified apps that have chosen not to cooperate with the energy manager will ignore these signals, and it is safe for any app to do so—except for the fact that cooperative mechanism will be followed by imperative ones if the app's energy consumption remains at odds with the energy manager's policy. Once we gain more experience with Jouler, we may consider delivering more information along with cooperative signals—such as the apps' current energy consumption rate and the energy manager's target—if cooperative apps find additional information useful.

Cooperative apps may connect cooperative signals to app-specific choices affecting energy consumption. For example, an email client may reduce the number of folders that are periodically synchronized and a browser may request lower-quality content. Because Jouler's cooperative signals directly reflect a user's energy manager's policies, they are much more powerful than ad-hoc triggers—such as low battery level—at enabling app energy awareness. A user may want an infrequently-used app to always limit its energy consumption and a frequently-used app to never limit its energy consumption, regardless of the current battery level.

Cooperative library Apps are free to respond to cooperative signals directly, but there are also a set of energy-aware design patterns common across many apps. To further encourage apps to collaborate with the energy manager, Jouler includes a library of cooperative mechanisms driven by its cooperative signals.

– **Energy-adaptive timers.** Background operations performed by smartphone apps are often driven by timers. For example, an email client may periodically contact a server to check for new mail. Unfortunately, the energy consumption resulting from a static rate may not be appropriate for all users or in all scenarios.
  Jouler provides *energy-adaptive timers* that adjust their firing rate in response to cooperative signals. Apps configure a maximum and minimum firing rate and step size when initializing the energy-adaptive timer. When they receive the `Reduce` signal, energy-adaptive timers reduce their firing rate by one step until they reach the minimum; when they receive the `Increase` signal, they increase their firing rate by one step until they reach the maximum.
  Using adaptive timers is easy. Developers can simply replace calls to existing timer interfaces with the new adaptive timers provided by Jouler.
– **Energy-delayed tasks.** Some tasks performed by smartphone apps are delay tolerant and can be deferred until conditions are favorable. For example, a music client may only download requested music to add to a local cache when an energy-efficient Wifi network is available and not over 4G. Unfortunately, it is difficult for apps to determine under what conditions tasks should be delayed.
  Jouler's provides *energy-delayed tasks* that use cooperative signals to determine when to run. Apps register an energy-delayed task with a maximum delay. The task will not run until the app receives the `OK` or `Increase` signal or the maximum delay is reached.
  Using energy-delayed tasks is also easy. Assuming that developers already have their task in a separate module so that it can be deferred, they need only to wrap the task in the new energy-delayed task object provided by Jouler.

We do not claim either of these mechanisms to be novel, and Jouler's cooperative library borrows freely from previous systems, such as Eon, which also adjusted timer rates to control sensor network node energy consumption [19]. However, to our knowledge Jouler is the first system to integrate these approaches into a smartphone energy management framework. The cooperative library also allows apps to factor out complicated and error-prone decision making concerning when to conserve energy, and instead focus on responding effectively to signals issued by the energy manager.

3.4 Imperative Mechanisms

Jouler's cooperative features encourage apps to manage their own energy consumption. However, there are many cases where cooperation will fail. First, as Jouler is introduced, most apps will not have been modified to cooperate with the energy manager. Second, some apps may not want to cooperate, either to selfishly gain performance or to maliciously waste energy. Finally, an app's attempts to cooperate may be insufficient to meet the energy manager's and user's goals. These limitations require that Jouler provide imperative mechanisms that force—rather than ask—apps to reduce their energy consumption. Imperative mechanisms both ensure that energy managers can control all apps while also encouraging apps to cooperate with the energy manager to manage their energy consumption more intelligently.

Jouler ties all imperative mechanisms to cooperative signals. Any time an energy manager applies an imperative mechanism to an app it is also sent the `Reduce` signal.

When the imperative mechanism is removed, the app is sent the `Increase` signal—but only after a delay to avoid feedback loops. Coupling cooperative signals to imperative mechanisms allows cooperative apps to continue to attempt to adjust their energy consumption while imperative mechanisms are applied.

To enable effective energy management policies, Jouler provides energy managers access to as many imperative mechanisms as possible to limit overall and per-app energy consumption. Most of these mechanisms other than screen brightness cannot be accessed by unprivileged apps in general.

Jouler currently provides energy managers with the following imperative mechanisms reflecting energy-saving features available on current smartphones:

- **CPU tuning.** Energy managers can change CPU governors and adjust the CPU frequency of dynamic voltage and frequency scaled (DVFS) processors by selecting either performance-boosting higher frequency or energy-efficient lower frequency.
- **App priorities.** Energy managers can set app scheduling priorities which affect the relative performance of multiple running apps. For example, an unthrottled app may achieve equivalent performance at a lower CPU frequency if its priority is increased relative to a throttled app.
- **Bandwidth throttling.** Energy managers can control per-app and global usage of available network interfaces.
- **Brightness adjustment.** Energy managers can control per-app and global screen brightness.

To allow energy managers to apply per-app policies, Jouler delivers a signal to the energy manager each time any app comes to the foreground. Energy managers can use this signal to adjust global settings such as the CPU frequency on a per-app basis while also enforcing background settings. Energy managers may also want to adjust per-app settings such as priorities to distinguish between foreground and background operation.

In some cases imperative mechanisms may have varied effects on apps. For example, slowing the CPU frequency may cause certain apps to consume more energy due to other components being active for a longer period of time. These complicated app interactions argue for the increased policy flexibility provided by Jouler, particularly given that default platform policies frequently ignore these complexities.

Intentionally omitted imperative mechanisms. Because Jouler's goal is to enable flexible and effective energy management of apps that users want to continue using, it does not allow energy managers to uninstall apps or kill app services, which can cause apps to misbehave. Jouler energy managers are free to suggest these actions to users if they could be beneficial.

3.5 Privacy Concerns

To manage energy effectively, energy managers have access to information about the apps running on their smartphone that some users may prefer not to reveal. Currently, Jouler uses a single permission mechanism to inform users of this risk during installation, but we are exploring more fine-grained permissions that could allow users to anonymize the app information provided to energy managers. This would affect policies that rely on identifying apps, but might alleviate some privacy concerns.

4 Example Energy Managers

Jouler is designed to allow flexible and innovative energy management policies to be implemented as energy managers. In this section, we describe few such policies.

Lifetime Targeting. Many previous approaches to energy management focus on meeting a target lifetime. By monitoring energy consumption and the remaining battery level, the energy manager can determine whether the user's lifetime target will be met. If their smartphone may run out of energy too soon, the energy manager can decide to use Jouler mechanisms in a way that reduces energy consumption with minimal performance degradation. On the other hand, if their smartphone may run for hours more than the expected target lifetime, due to less usage or short term unexpected charging sessions in between, then the manager can use Jouler mechanisms to boost performance for better user experience.

While lifetime targeting is conceptually simple, dynamically deciding the trade off between conserving energy and boosting performance is difficult. It is also hard to predict hours before if the target can be met or not due to possible fluctuations in app usage. Although we are not sure what lifetime targeting approach will prove most effective, by enabling the distribution and testing of new approaches Jouler accelerates the process of developing effective solutions.

App Based Throttling. Smartphone users generally install a large number of apps over the time they use their device, some of which they use regularly and are clear favorites—such as a default email client, browser, messaging app and social networking client. There are also those apps which a user has installed but has hardly used or would not care if there is a slight performance degradation to reduce energy consumption. An energy manager can allocate a major chunk of energy to the regular apps and monitor the energy consumed by most favorite apps and least favorite apps. If the non-favorite apps consume more energy than desired, the energy manager can start throttling them to allow the user to access the regular apps for longer period. By observing users' app usage over a period of time, the energy manager may be able to suggest what apps should be on their list of favorites.

Reward Efficient Content Delivery. To determine appropriate per-user settings the energy manager needs better understanding of app energy consumption. A common challenge faced by many energy management approaches is distinguishing between two apps: one that uses a great deal of energy because it is poorly written, and a second that inherently needs a great deal of energy to function properly. Without more information about what the apps are doing, these two very different apps are indistinguishable. Moreover app background energy consumption varies widely between apps, and between the same app used by different users. Whether legitimate or not, the variation in background energy consumption weakens the connection between how much the smartphone is used and how much energy it consumes.

For this reason, Jouler provides energy managers with app usage and interaction information. An energy manager could combine energy consumption with the amount of data delivered through the screen and audio port to determine how efficiently the

app is delivering content to the user. In the example above, this would allow the energy manager to distinguish between a streaming video client (inherently-high consumption) and a poorly-written chat client (buggy consumption). This will also help the manager to determine whether the energy consumed for background work is required or is a waste. We anticipate that the Jouler framework will lead to more intelligent energy management policies that observe a variety of aspects of app and user behavior.

5 Implementation

We have implemented the design, we just discussed, by modifying the Android Open Source Platform (AOSP) version 4.4.4 named KitKat. The detailed information about app usage and energy consumption are collected by a lightweight privileged service `JoulerPolicyService` running in the platform from boot time. This information includes overall, per-app, per-component, foreground and background breakdown of usage and energy consumption. These inputs can be accessed by the energy managers through the custom apis `JoulerStatistics` and `JoulerPolicy`. The later api also provides an interface for using all the imperative mechanisms discussed in Jouler design. For example, AOSP in LG Nexus 5 uses the `online` governor as a default CPU governor which jumps between low and high CPU frequencies based on predetermined workload thresholds. Using Jouler, an energy manager, if it chooses, can select the `userspace` governor and set the highest frequency to boost performance of a heavyweight app which is also a favorite of the user. But in order to access Jouler's framework, the energy manager apps need to use the new `CAN_MANAGE_ENERGY` android permission. For stability reasons, Jouler allows only one energy manager to run at any given time even if multiple energy managers are installed in a single device.

The cooperative signals and mechanisms are also implemented in a manner that is intuitive and easy-to-use for existing community of app developers. The easiest way to send signals in an Android environment is to broadcast intents. So, we defined a new intent `ACTION_ENERGY_ALERT` having three separate lists of package names of installed apps, each list corresponds to apps who need to reduce energy consumption, or apps who are doing okay or apps who can boost their performance as they are below the threshold determined by a particular energy manager. Cooperating apps only have to register to listen to this broadcast intent and decide whether they should reduce their energy consumption or not. For other cooperative mechanisms like adaptive timers, we have stayed true to the current AlarmManager implementation in Android. We added a new `setAdaptive` method to the existing android.app.AlarmManager that is similar to existing methods like setInexactRepeating but accepts an extra input to determine the longest deadline till which the work can be delayed if needed. On the other hand the wrapper for delayed task is found in the Jouler api we have mentioned earlier.

Overall our experience of implementing Android Jouler suggests that it should be straightforward to implement Jouler for other smartphone platforms, allowing cross-platform distribution of effective energy management policies. Jouler's cooperative library will need to be reimplemented, but the service mainly exposes statistics and control mechanisms provided by operating systems for decades. However, without access to sources for iOS or Windows Mobile we can only speculate about the development burden on these other smartphone platforms.

5.1 Energy Manager Implementation

We implemented and distributed three simple and straightforward energy managers:

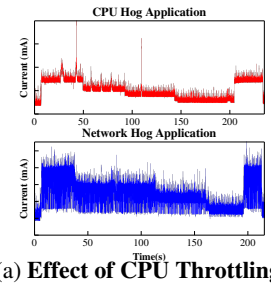
- The **Favorites Manager** allows users to select a list of favorite apps. Periodically, it compares the total energy consumption of the favorite apps and that of the other apps. If the later is higher, then the energy manager restricts network usage by the non-favorite apps when they run in the background and reduces brightness to reduce screen energy consumption when they run in the foreground.
- The **Blacklist Manager** is identical to the *Favorites* manager except that it asks users to choose the apps they like the least. Accordingly, it tries to reduce energy consumption by the blacklisted apps when they run in the screen foreground or background.
- The **Lifetime Manager** attempts to achieve at least the target lifetime hours configured by the user. With every alternate battery level drop, the manager compares the current battery discharge rate with the expected discharge rate. Accordingly, the manager gradually throttles the CPU, app priorities, network usage and screen brightness. If it is still unable to reach the target, it notifies the user how many hours left before the device runs out of energy. Currently the energy manager does nothing if the achieved lifetime is much more than what the user has asked for.

6 Evaluation

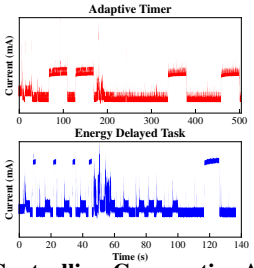
Our evaluation of the Android prototype demonstrates that Jouler provides both effective and flexible energy management policies. We evaluate Jouler in two steps. First we use energy benchmarks to show that Jouler's mechanisms are effective. Second, we perform a ten day deployment of Jouler's platform modifications and of three simple but different energy managers on the PHONELAB testbed.

6.1 Energy Benchmark

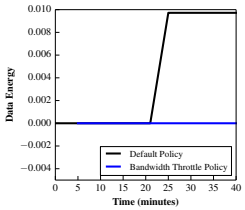
First, we wanted to test if Jouler's privileged system service, running continuously in the platform to collect detailed app usage and energy consumption information, causes any overhead. We fully charged two LG Nexus 5 smartphones, flashed a clean Android build on one of them and flashed an image with Jouler modifications on the other. We kept these phones unplugged with display screen off for 8 hours. The same battery level drop for both phones assured us that there is no perceptible overhead for Jouler. Next, we wrote a simple Android energy benchmark which is configurable to hog the processor and network either continuously in the screen foreground or periodically in the background. We tested the imperative mechanisms using this benchmark and some of those results are presented in this section. To test the cooperative mechanisms, we wrote one client app which uses both adaptive timer and energy delayed task wrappers to cooperate with our simple energy manager which can send *Reduce*, *OK* and *Increase* signals to the client app.



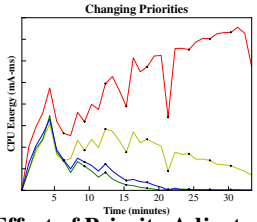
(a) **Effect of CPU Throttling.**



(b) **Controlling Cooperative Apps.**



(c) **Effect of Bandwidth Throttling.**



(d) **Effect of Priority Adjustment.**

Fig. 3: **Effects of few Jouler Mechanisms on Energy**

6.2 Jouler Mechanisms

To verify that Jouler's mechanisms were having the intended effect on our benchmark, we measured the current output of a Samsung Galaxy Nexus smartphone using a Monsoon Power Monitor [2]. Figure 3a shows the effect of reducing the processor frequency in steps when the benchmark is hogging both the CPU and network. As expected, CPU throttling reduces the power consumption of both hogs.

Figure 3d shows the effect of adjusting the Linux priorities of four instances of our energy benchmark running as CPU hogs and competing together for the processor. As the priorities of two hogs are raised—the yellow hog to the highest priority and the red hog to an intermediate level—their share increases, and the shares of the two other hogs are decreased as their priorities are lowered.

It is important that Jouler's imperative mechanisms do not force existing, non-cooperative apps to misbehave. In a Nexus 5 with pre-installed apps, we ran a whitelist energy manager that throttles bandwidth of all non-favorite apps running in the screen background. Figure 3c shows that bandwidth throttling saves power consumption by limiting Gmail background energy consumption. During the experiment, we verified that Gmail continued to behave normally and did not crash, confirming our expectation that well-written apps can handle resource limitations.

Our evaluation of Jouler's cooperative mechanisms is shown in Figure 3b. The power monitor output confirms that Jouler's cooperative mechanisms work as expected, slowing the energy-aware timer and stopping energy-delayed tasks when receiving the *Reduce* signal and restarting energy-delayed tasks when receiving the *Increase* signal.

6.3 Deployment

Our final step of evaluation consists of distributing platform modifications for implementing Jouler and an integrated app which provides a selection of energy managers

Preferred Energy Manager	Users
`Lifetime Energy Manager`	44
`Favorites Energy Manager`	31
`Blacklist Energy Manager`	10
`Default Android Energy Manager`	88

Table 1: **Energy Managers preferred by Users.**

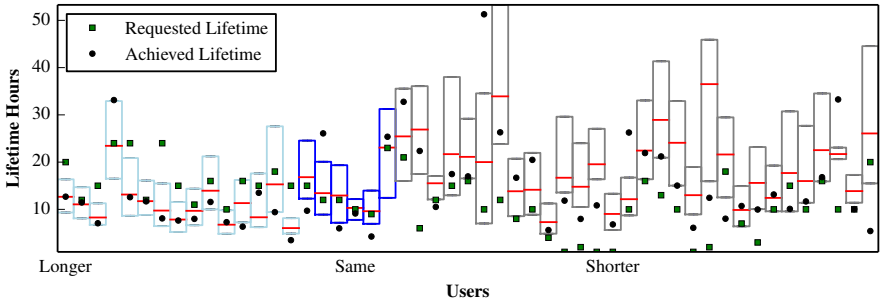


Fig. 4: **Lifetime Energy Manager Comparison.** The red line shows the median while the upper and the lower edges of each box shows upper and lower quartiles of the expected lifetime hours distribution for each user. The different colored boxes labeled as longer, same and shorter signify users requesting lifetime hours greater than, equal to or slightly less than and much lesser than the median respectively. **This analysis aims to compare the expected, requested and achieved lifetimes for each user.**

to the participants of the PHONELAB testbed as an over-the-air update. The integrated app offers 4 choices to users - Lifetime Energy Manager, Favorites Energy Manager, Blacklist Energy Manager and No Energy Manager. The last choice allows users to continue using or go back to the default Android energy management. These energy managers are chosen because they are simple to use and understand. Users can also switch between energy managers multiple times. We did not advertise or try to influence the testbed participants to use the app or configure it in a particular way. This is done to evaluate if the energy managers are intuitive and easy to understand and use. 203 participants received the update and 173 participants access the app at least once. Table 1 shows the breakdown of users who used one energy manager for the longest period during the experiment lifetime. After 10 days a short survey was distributed to collect their overall impressions of the Jouler system. PHONELAB's built-in platform instrumentation and logging capabilities were used to collect data generated by the platform and energy managers. The entire experiment was reviewed and approved by the University at Buffalo's Institutional Review Board (IRB).

Lifetime Energy Manager. To understand the lifetime expectations of different users, first we need to know for how long can a device stay off the plug on any given day. We have recorded battery related details for all PHONELAB participants since September 2014. Using this data, we computed the average battery drain per hour for each user and the expected lifetime hours based on that rate. In Figure 4 for all the participants who selected the lifetime manager we show the distribution of expected lifetime hours. Users did not always request lifetime hours close to what is expected of their device. So we

Rank	App Name	Occurrences
1	Google Play Newsstand	10
1	Google Play Books	10
2	Google Play Games	9
2	Android Movie Studio	9
3	Google Play Music	8
3	Google Wallet	8
4	Superuser	7
5	Earth	6
5	News & Weather	6
5	Google	6

(a) **Most Common Blacklisted Apps.**

Rank	App Name	Occurrences
1	Chrome	17
2	Facebook	12
2	Hangouts	12
2	Dialer	12
2	Camera	12
3	Youtube	11
3	Maps	11
4	Gallery	10
4	Clock	10
5	Gmail	9

(b) **Most Common Favorite Apps.**

Table 2: **Configuring Blacklist and Favorites Energy Managers.**

group the users based on whether the requested lifetime hours is shorter, longer or equal to the lifetime hours they usually experience. It needs to be pointed out that the energy manager has a constraint that does not allow users to select a lifetime goal beyond 24 hours. The energy manager failed badly for most of the users in the first group, who requested a comparatively longer lifetime. Thus, our *Lifetime* energy manager is not a good fit for users running heavyweight apps frequently or having heavy device usage because this manager cannot decrease the discharge rate by only following global policies. Rather *Blacklist* or *Favorites* energy manager can be a better choice in these cases. The energy manager fared slightly better for the group of users who requested similar lifetime hours. In the last group, a large number of users requested a shorter lifetime either due to the constraint imposed by the energy manager or because they do not care about very long lifetime hours. In these cases, a good energy management policy will be one which can boost performance instead of being too conservative about saving energy while reaching the lifetime target. Though our energy manager did not fare well across all categories of users, it provided us with interesting insights to improve lifetime management policies in future.

Blacklist and Favorites Energy Manager. The other two energy managers we distributed are app based. At the beginning, users, who selected one of these managers, are prompted to choose one or more apps to put in the blacklist or whitelist for *Blacklist* and *Favorites* energy managers respectively. In Table 2 we found pre-installed google apps to be most commonly blacklisted. On the other hand, users were more likely to select browsers and social media apps as favorites.

Survey. Continuing with the evaluation, we distributed a short survey among the participants. Our goal is to determine if users are interested in using energy manager apps in the future. We asked users if they found understanding and configuring our energy managers easy and if they appreciated having more control over how energy is managed in their devices. We asked users to list which energy manager they preferred the most and to state problems they faced while using any of the energy managers. We also asked for any suggestion they might have to improve our energy managers. 88 participants responded, 80% of whom appreciated having this extra control over energy

management. But 25% of the participants reported facing problems while the energy manager was running. The most common complaint was about the harsh decrease in the display screen's brightness by the energy managers which hampered user-experience. Some users did not find the energy managers to be perceptibly effective. Many users also pointed out lack of instructions from our end to be a key factor for not knowing how to use the app, which might be the reason why a majority of the participants did not use any of the three energy managers. Some also suggested instead of their having to select their favorite apps manually, it would be more helpful if the energy manager could internally decide which are the preferred apps from usage related information.

7 Related Work

We divide related work into projects related to Jouler's inputs, its mechanisms, its policies, before briefly discussing other Android apps that attempt to help users manage their smartphone's energy consumption. While some portions of Jouler draw on similar work in the sensor network and mobile systems areas, to our knowledge no existing system provides the capabilities and flexibility of Jouler.

Jouler Inputs Effective energy management relies on accurate energy measurement and attribution. Previous tools such as PowerTutor [24] have demonstrated model-driven approaches to determining per-component and per-app energy usage, with this approach being largely replicated by Android's internal Fuel Gauge component. Because many modeling approaches struggle with limited visibility of aggregate energy consumption, VEdge [20] uses only measurement of the battery voltage to infer current draw and therefore energy consumption. Other recent projects have provided improved approaches incorporating temporal variation in exogenous factors such as network signal strength into models to make them more accurate [5]. Jouler's energy managers will benefit from future improvements in this area.

Jouler Mechanisms Multiple systems have attempted to establish new operating system energy management mechanisms or encourage more adaptive app energy consumption. ECOSystem proposed a system where each process was given an energy budget to spend and would use this "currentcy" to schedule tasks [23]. Both Pixie [11] and Eon [19] provided the ability for sensor network programs to adapt to changing energy availability, but did so by relying on special languages or program structure that would be infeasible to apply to smartphones. Odyssey [7] focuses largely on enabling per-app resource adaptation, a capability complementary to Jouler. Recently the Cinder [18] OS based on HiStar [22] proposed new mechanisms enabling explicit resource allocation and accounting which would help Jouler control uncooperative apps. The battery virtualization proposed by PowerVisor [25] would also be useful as a Jouler mechanism but does not by itself address the policy problem.

Jouler Energy Managers Recent work on smartphone energy management has used measurements from large user communities to categorizing apps based on energy consumption. Carat has been installed by over $500{,}000$ devices, and attempts to identify

two energy anomalies: *bugs* and *hogs* [14]. Carat is a notable attempt to address smartphone energy management but suffers from several drawbacks that Jouler could help address. First, Carat's app generalizations fail to consider the differences in smartphone users our data has demonstrated, which render an app that one user considers acceptable a hog to another user. Second, Carat has the same heavy-handed mechanisms available to it as all other current energy management approaches: remove the app or stop using it. Jouler's mechanisms would provide Carat with more tools to enforce its classification. PowerLet [8] is another system that suffers a similar weakness in that it relies on users to take energy saving actions, rather than creating an interface as Jouler does which allow these actions to be performed programmatically.

Existing Energy Management Apps The Google Play Store provides Android users with multiple options for controlling their energy consumption but many of them require root privileges. JuiceDefender[2] controls the underlying smartphone hardware such as enabling and disabling wireless interfaces to attempt to keep energy consumption under control. Easy Battery Saver[3] offers users the choice of multiple energy modes and a variety of battery lifetime estimation tools. Unfortunately, both these apps have to apply policies across the entire phone and cannot control individual apps. The mechanisms currently available to these tools are too blunt to effectively control apps with varying usage patterns.But we expect that such approaches may be more effective with the additional app information and fine-grained mechanisms Jouler provides. Tools such as Mr. Nice Guy[4] which allows per-app priority adjustments, force users to manually fiddle with priorities and act as energy managers to implement specific policy goals.

8 Future Work and Conclusions

To conclude we have presented the Jouler policy framework which enables flexible and effective smartphone energy management benefiting both developers and end users. With the Jouler service running on the PHONELAB testbed, we are planning several next steps. Based on the lessons learned from our evaluations, we plan to improve our energy managers by having policies that do not hinder user experience. For example, brightness level needs to be changed more intuitively. We also plan to add new mechanisms to the existing framework that allow the energy manager to effectively enhance device performance for users who do not mind shorter lifetime hours. Lastly, we are working to modify several apps with sources available as part of the AOSP to allow them to use Jouler's cooperative library. We expect that the continued evaluation on the PHONELAB testbed will lead to new results and hope to eventually prepare a patch allowing Jouler to be considered for inclusion in the AOSP.

References

[1] Battery Life: Is That All There Is? . `http://www.jdpower.com/resource/jd-power-insights-i-battery-life-all-there`.

[2] http://www.juicedefender.com

[3] http://goo.gl/GfcI2q

[4] http://goo.gl/8utSxe

[2] Monsoon power monitor. http://www.msoon.com/LabEquipment/PowerMonitor/.

[3] BANERJEE, N., RAHMATI, A., CORNER, M., ROLLINS, S., AND ZHONG, L. Users and batteries: interactions and adaptive energy management in mobile systems. In In Proc. UbiComp, 2007. 11 David R. Choffnes and Fabin (2008).

[4] BROUWERS, N., ZUNIGA, M., AND LANGENDOEN, K. Neat: a novel energy analysis toolkit for free-roaming smartphones. In Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems (2014), ACM, pp. 16–30.

[5] DING, N., WAGNER, D., CHEN, X., PATHAK, A., HU, Y. C., AND RICE, A. Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. In Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems (New York, NY, USA, 2013), SIGMETRICS '13, ACM, pp. 29–40.

[6] DONG, M., CHOI, Y.-S. K., AND ZHONG, L. Power modeling of graphical user interfaces on oled displays. In Proceedings of the 46th Annual Design Automation Conference (New York, NY, USA, 2009), DAC '09, ACM, pp. 652–657.

[7] FLINN, J., AND SATYANARAYANAN, M. Energy-aware adaptation for mobile applications. SIGOPS Oper. Syst. Rev. 33, 5 (1999), 48–63.

[8] JUNG, W., CHON, Y., KIM, D., AND CHA, H. Powerlet: An active battery interface for smartphones. In Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing (New York, NY, USA, 2014), UbiComp '14, ACM, pp. 45–56.

[9] LIN, F. X., WANG, Z., AND ZHONG, L. K2: A mobile operating system for heterogeneous coherence domains. In Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (New York, NY, USA, 2014), ASPLOS '14, ACM, pp. 285–300.

[10] LIU, J., PRIYANTHA, B., HART, T., RAMOS, H. S., LOUREIRO, A. A., AND WANG, Q. Energy efficient gps sensing with cloud offloading. In Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (2012), ACM, pp. 85–98.

[11] LORINCZ, K., RONG CHEN, B., WATERMAN, J., WERNER-ALLEN, G., AND WELSH, M. Resource Aware Programming in the Pixie OS. In ACM Conference on Embedded Networked Sensor Systems (SenSys'08) (November 2008).

[12] MITTAL, R., KANSAL, A., AND CHANDRA, R. Empowering developers to estimate app energy consumption. In Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (New York, NY, USA, 2012), Mobicom '12, ACM, pp. 317–328.

[13] NANDUGUDI, A., MAITI, A., KI, T., BULUT, F., DEMIRBAS, M., KOSAR, T., QIAO, C., KO, S. Y., AND CHALLEN, G. Phonelab: A large programmable smartphone testbed. In Proc. 1st International Workshop on Sensing and Big Data Mining (SenseMine 2013) (November 2013).

[14] OLINER, A. J., IYER, A. P., STOICA, I., LAGERSPETZ, E., AND TARKOMA, S. Carat: collaborative energy diagnosis for mobile devices. In SenSys (2013), C. Petrioli, L. P. Cox, and K. Whitehouse, Eds., ACM, p. 10.

[15] PUNZALAN, R. Smartphone Battery Life a Critical Factor for Customer Satisfaction . http://www.brighthand.com/default.asp?newsID=18721.

[16] QIAN, F., SEN, S., AND SPATSCHECK, O. Characterizing resource usage for mobile web browsing. In Proceedings of the 12th annual international conference on Mobile systems, applications, and services (2014), ACM, pp. 218–231.

[17] RAVINDRANATH, L., AGARWAL, S., PADHYE, J., AND RIEDERER, C. Procrastinator: Pacing mobile apps usage of the network. In Proceedings of the 12th annual international conference on Mobile systems, applications, and services (2014), ACM, pp. 232–244.

[18] RUMBLE, S. M., STUTSMAN, R., LEVIS, P., MAZIÈRES, D., AND ZELDOVICH, N. Apprehending joule thieves with cinder. In MobiHeld '09: Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds (New York, NY, USA, 2009), ACM, pp. 49–54.

[19] SORBER, J., KOSTADINOV, A., BRENNAN, M., GARBER, M., CORNER, M., AND BERGER, E. D. Eon: A Language and Runtime System for Perpetual Systems. In ACM Conference on Embedded Networked Sensor Systems (SenSys'07) (November 2007).

[20] XU, F., LIU, Y., LI, Q., AND ZHANG, Y. V-edge: Fast self-constructive power modeling of smartphones based on battery voltage dynamics. In Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (Berkeley, CA, USA, 2013), nsdi'13, USENIX Association, pp. 43–56.

[21] XU, F., LIU, Y., MOSCIBRODA, T., CHANDRA, R., JIN, L., ZHANG, Y., AND LI, Q. Optimizing background email sync on smartphones. In Proceeding of the 11th annual international conference on Mobile systems, applications, and services (2013), ACM, pp. 55–68.

[22] ZELDOVICH, N., BOYD-WICKIZER, S., KOHLER, E., AND MAZIÈRES, D. Making information flow explicit in histar. In Proceedings of the 7th symposium on Operating systems design and implementation (2006), USENIX Association, pp. 263–278.

[23] ZENG, H., FAN, X., ELLIS, C. S., LEBECK, A., AND VAHDAT, A. ECOSystem: Managing Energy as a First Class Operating System Resource. In Proc. Architectural Support for Programming Languages and Operating Systems (ASPLOS) (San Jose, CA, October 2002).

[24] ZHANG, L., TIWANA, B., QIAN, Z., WANG, Z., DICK, R. P., MAO, Z. M., AND YANG, L. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (New York, NY, USA, 2010), CODES/ISSS '10, ACM, pp. 105–114.

[25] ZHANG, N., RAMANATHAN, P., KIM, K.-H., AND BANERJEE, S. Powervisor: A battery virtualization scheme for smartphones. In Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services (New York, NY, USA, 2012), MCS '12, ACM, pp. 37–44.