CSR: Small: Jouler: A Cross-Device Application Energy Management Framework for Smartphones

NSF Proposal
14 Jan 2014

PI: Geoffrey Challen (*University at Buffalo*

With the usage of energy-constrained smartphones exploding, smartphone energy management has received an increasing amount of attention. Researchers have succeeded in developing new energy models, measurement tools, and more efficient implementation of common primitives such as localization. But together, these advances have failed to fully address the problem, with users reporting battery lifetime as not only their top concern with today's smartphones, but a worsening concern. Already unsatisfied with device lifetimes, smartphone users are less likely to explore the growing number of innovative apps, so if the full potential of smartphone devices is to be realized, more innovation in energy management is needed.

This project proposes a new cross-device application energy management framework for Android smartphones called Jouler. Jouler has three goals that distinguish it from current approaches. First, Jouler is *automated*, controlling app energy consumption without requiring user input. Removing the human from the loop requires Jouler to be able to distinguish between high app energy consumption caused by legitimate app requirements from that caused by software bugs. Second, Jouler is *flexible*, encapsulating energy management strategies in policy modules that users can assign to each app. Data collected on the PhoneLab testbed shows significant differences in app usage, charging patterns, and battery lifetime expectations between participants, and policy modules give Jouler the flexibility required to support this variation. Third, Jouler is *cross-device*, allowing full integration into the Android platform and deployment on all Android smartphones. Because hardware energy management capabilities different significantly across components, Jouler must provide a new abstraction layer to ensure that policy modules are portable across multiple devices integrating different hardware components. To evaluate the ability of Jouler to improve smartphone energy management, it will be integrated into the Android platform and tested on the PhoneLab smartphone testbed, the only public testbed able to distribute platform updates.

Keywords: energy management; **mobile systems**; **smartphones**.

Intellectual Merit: Realizing Jouler requires addressing two significant open research challenges. First, while measuring app energy consumption has received a great deal of attention, putting consumption into context has not. Consumption alone may provide helpful feedback to users, but is not sufficient to enable effective policies, since the behavior and efficiency of different apps varies, as does their importance to each user. These uncertainties make energy consumption alone usable for human interpretation, since users can use their own experience to put the numbers in context, but *unusable* for automated response or app energy prioritization, one of the goals of Jouler. Second, tuning energy consumption currently requires deep integration with the specifics of each particular hardware component, making writing cross-device policies impossible. A policy that relies on a certain feature of a Wifi chipset available on a particular device will fail on a second device if its Wifi driver fails to support the feature. Without a narrow waist enabling cross-device policies, energy management improvements achieved on a given device will quickly become obsolete given the rapid improvements in smartphone technologies. Jouler also provides novel software control mechanism enabling coordinated energy management with modified apps.

Broader Impacts: The proposed broader impact activities will both train and excite the next generation of computer scientists in energy management strategies for smartphones. First, a new graduate course on platform energy management will be developed and taught by the PI, broadening the participation of students in studying energy consumption and providing students with a unique opportunity to modify the Android platform. Second, once the Jouler policy module framework is developed, a yearly open competition will be held to drive interest in policy module development and help establish the effectiveness of the Jouler framework. Finally, we propose to integrate energy management into `ops-class.org`, a free online educational environment created to train systems programmers.

INTELLECTUAL MERIT

Smartphone energy management remains important, with users reporting battery lifetime as a top and worsening concern with today's smartphones [19] and rapidly downloading task management and energy monitoring [18] tools from app marketplaces. Clearly, short device lifetimes continue to frustrate users, and this frustration is significant because it is likely preventing many users from exploring the wide variety of apps available that continue to unlock the transformative power of mobile smartphones. If we are truly to see the potential of the over one billion smartphones deployed worldwide and all of the emerging innovative apps this enormous distributed system will support, this energy bottleneck must be addressed.

Fortunately, the research community has responded to this challenge by continuing to improve the modeling [36, 33] and measurement [18] of smartphone app energy consumption, while also developing many techniques to reduce overall device energy consumption through software and hardware improvements. Unfortunately, continued user frustration and concern with smartphone lifetimes indicates that the problem is far from solved, and smartphone platforms still provide little to no support for managing and controlling app energy usage. On Android, it is impossible for apps to even access the same energy modeling information made available to users through the Fuel Gauge, and few to no platform interfaces are provided to aid the process of building energy-aware apps.

The non-existence of platform-level energy management is due to three open challenges unaddressed by previous research efforts. First, given the wide variety of apps with different behavior and requirements, app energy consumption[1] information cannot be used on its own to make automated energy management decisions. Second, the diversity of smartphone hardware components, each with their own energy management capabilities and interfaces, makes it difficult to generalize approaches that work on one device to other devices. Finally, the diversity of available apps, app usage patterns, and user charging habits means that there is no one canonical smartphone app energy management problem to solve, and that monolithic solutions able to satisfy one type of smartphone user will frustrate most. In addition, while app marketplaces and app distribution platforms have accelerated smartphone research that can be performed at the app level, no equivalent distribution channel exists for changes to even open-source smartphone platform software such as Android, making it impossible for most researchers to distribute the platform changes needed to manage app energy usage to a broad audience of representative smartphone users.

We propose to address these three open energy management problems through the Jouler cross-device energy management framework, which will be integrated directly into the Android Open Source Platform (AOSP). Allowing Jouler to be *automated* requires collecting more information about app behavior in order to allow our framework to distinguish between apps that are wasteful and those that simply require more energy to function. To provide a *cross-device* framework, Jouler wraps hardware components in platform-level drivers each providing a simplified interface. And Jouler's design cleanly separates energy usage policies—implemented by small pieces of code called policy modules—from the mechanisms used to control app energy usage, giving Jouler the *flexibility* needed to support the huge diversity in smartphone users and apps. A critical enabler of this project is the PHONELAB smartphone testbed, build and operated by our group, which provides the unique capability to distribute Android platform updates to a large number of representative smartphone users, a capability required to test and validate the Jouler framework.

The rest of this proposal is structured as follows. We motivate Jouler's design through an energy usage experiment distributed to 78 PHONELAB participants in Section 1, which also demonstrates our ability to safely and effectively distribute Android platform changes to a group of representative smartphone users. Section 2 describes the design of the Jouler framework in detail, including proposed methods of determining app utility and putting energy consumption into context (§ 2.2), mechanisms for limiting app energy consumption by controlling hardware (§ 2.3) and software (§ 2.4), and the policy module architecture that allows Jouler to evolve and remain flexible as app and user needs change (§ 2.5). We continue by discussing related work (§ 3), the broader impact components of our proposal (§ 4), and conclude with our qualifications and a preliminary task plan (§ 5).

[1]To avoid confusion when discussing both energy and user interaction through the paper, we use *consumption* exclusively when referring to energy and *usage* exclusively when referring to interaction.

1 — MOTIVATION

By Gender			
Female	127	Male	122
By Age			
< 18	0	35–39	28
18–20	12	40–49	52
21–24	21	50–59	34
25–29	34	> 60	9
30–34	35		

Table 1: **PHONELAB demographic breakdown.**

To motivate the design of Jouler we begin by presenting data from an app usage experiment performed on the PHONELAB smartphone testbed. PHONELAB is a public smartphone testbed run by the PI at the University at Buffalo. 288 students, faculty, and staff carry instrumented Android Samsung Galaxy Nexus smartphones and received subsidized service in return for willingness to participate in experiments. Table 1 shows that PHONELAB participants are drawn from a wide variety of age brackets while also representing a balanced mixture of genders, making our results applicable to the broader population of over one billion smartphone users. Currently PHONELAB is open for public app-level experimentation, but is beginning to support platform experimentation as well. This is a critical features, because while experiment apps have had some success deploying on app marketplaces even when they provide no obvious user benefit [5], PHONELAB currently represents the only way for researchers to distribute experimental platform updates to a large number of representative users.

Collecting the fine-grained energy consumption and app behavior information required to formulate Jouler's design requirements required this unique PHONELAB capability. At present, Android lacks even an API allowing apps to receive energy breakdowns of their own consumption, much less energy consumed by other apps and the system as a whole, which could represent a security weakness. To conduct previous experiments [17] we used introspection to inspect the internals of the `BatteryStatsImpl` object used by the Android Fuel Gauge, but at fifteen-minute intervals which we considered too long to provide information detailed enough to support this project. In addition, key information about app usage of the display and audio that we needed to better understand app utility was missing.

Instead, we modified the platform itself to collect information about app energy consumption and content delivery. Logging was added to the `SurfaceFlinger` and `AudioFlinger` Android platform components to record usage of the screen and audio, and more significant changes were made to the *Activity Services* package to log transitions between activities and record energy consumption asynchronously after each transition. Table 2 lists and describes the main log tags that generated data for this study.

We deployed the platform update containing these changes on November 23rd, 2013, using a dedicated PHONELAB testbed management application to download and apply the update after prompting the participant to limit the impact of the reboot cycle required. Fairly quickly most of the participants received and applied the update, enabling the extra instrumentation and logging. Note that participation in this IRB-approved experiment also required participants to download and install an app collecting other data that can be observed from outside the platform. Installing the app indicated consent, and no logs generated by the platform were collected from any participant that did not also install the experiment app.

We performed a detailed measurement study of approximately one week's worth of data from 78 PHONELAB smartphones running the instrumented Android platform image. The dataset of 5.5 GB of compression log files represents 500 participant days during which 401 apps were used a total of 31,750 times for a total of 1100 hours of active use. This represents the first time that we utilized the ability of PHONELAB to distribute platform updates to participants to improve our visibility of platform behavior, and to our knowledge no other energy consumption study has collected data at this granularity. Next, we present several salient results from our study that motivate Jouler's design.

Tag	Count	Purpose
`ActivityTransition`	433,273	Logs app transitions between the background and foreground.
`ActivityEnergy`	634,385	Records app energy usage on foreground-background transition boundaries.
`Snapshot`	44,996,395	Snapshots app energy usage at 15 minute intervals.
`ScreenRefresh`	85,763,852	Logs every screen refresh.
`AudioAction`	749,028	Record audio start, stop and pause events.

Table 2: **Tags used by PHONELAB platform instrumentation.** The Android logging framework was used to extract fine-grained energy usage and app behavior information from the Android platform.

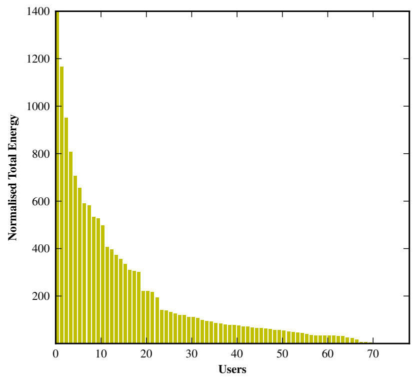


Figure 1: **Per-participant energy consumption variation.** Our data shows an enormous amount of energy consumption variation between participants, indicating significant differences in participant lifetime expectations and charging behavior.

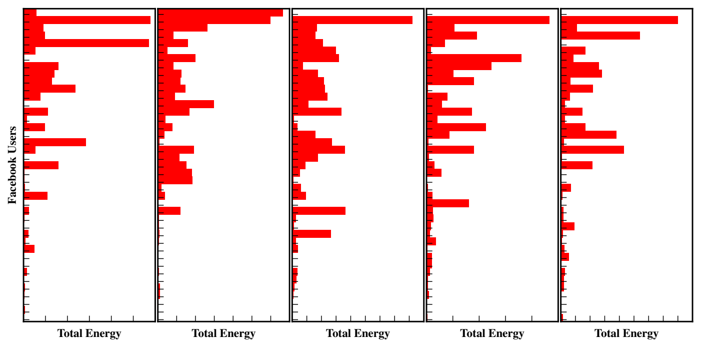


Figure 2: **Temporal energy consumption variation.** Energy consumed by the Facebook app is shown for a subset of participants. While some participants are consistent Facebook users, others show high amounts of temporal variation.

1.1 — Energy Consumption Varies Significantly Between Users

To begin, Figure 1 plots the normalized energy consumed by participants during the experiment and shows an enormous amount of variation between light and heavy users. Initially we were so surprised by these results that we conclude that these numbers must be flawed, and so we contacted the top several participants to help confirm our results. We were able to verify all of them as heavy users, and received several complaints about their short battery lifetimes, but could not identify any anomaly in this result. These participants also confirmed that they were charging frequently to accommodate their high consumption.

This level of variation has several important consequences for the design of Jouler. First, it demonstrates that there is no such thing as a normal battery lifetime across smartphone users. Some are used to frequently recharging their phones and others are used to their phones lasting for one day or more, and any effective energy management framework must be flexible enough to support both. Second, it hints at the variation in smartphone usage, with some users installing and using energy-hungry apps while others use their smartphones primarily almost as a cell phone. Third, it shows that app categorizations based on aggregate consumption are unhelpful, since the same app causing unacceptable drain for a light user may be acceptable to a heavy user. Jouler accommodates all this variation through policy modules (§ 2.5).

1.2 — Energy Consumption Varies Significantly Between Apps

Overall energy consumption variation between users is also accompanied by significant variation in app energy consumption. Table 3 shows the top and bottom five apps used by at least ten participants in the study ordered by the rate at which the app consumes energy, showing several orders of magnitude difference between the fastest and slowest consumers. Some of these aggregate results match expectations, such as the high consumption of Android Music and the flashlight app; others are more surprising, such as the low usage of Skype, which has been identified as an energy "hog" by previous work [18], but which may be used only for chat by the PHONELAB participants studied. Clearly it is difficult to impossible to make sense of these numbers without more information, which motivates Jouler's design that computes per-app utility values to put energy consumption into context (§ 2.2).

Rank	Consumption Rate (A)	Value
1	`Facebook Messenger`	0.774
2	`Google+`	0.614
3	`Super-Bright LED Flashlight`	0.600
4	`UB Parking`	0.598
5	`Android Music`	0.446
5	`Skype`	0.040
4	`YouTube`	0.036
3	`ESPN SportsCenter`	0.021
2	`The Weather Channel`	0.019
1	`Bank of America`	0.011

Table 3: **Fastest and slowest energy-consuming apps.**

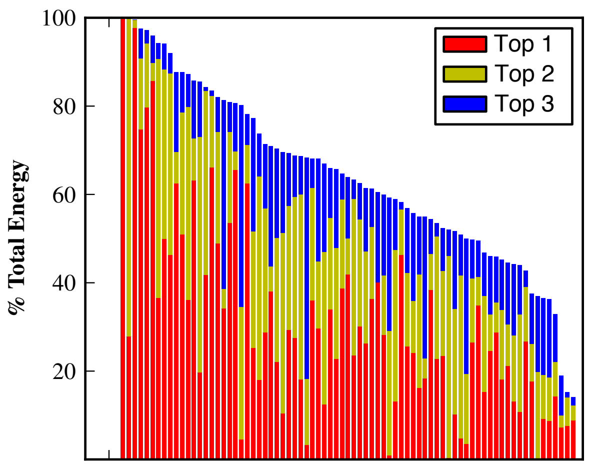


Figure 3: **Energy consumption by the top three most-used apps.** In most cases, consumption lines up with foreground time, implying that dramatic improvements in battery lifetime require removing popular apps.

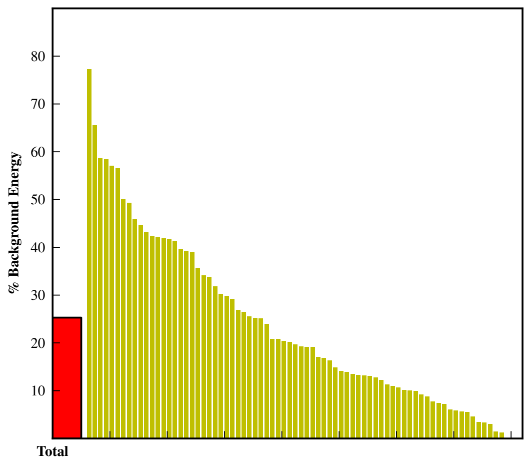


Figure 4: **Per-participant percentage of energy consumption consumed in the background.** A significant amount of energy is used by most apps in the background, leading to potential user confusion over poor battery lifetimes.

1.3 — Energy Consumption Varies Significantly Over Time

Overall energy consumption variation between users is also accompanied by per-user temporal variation in app energy consumption. Figure 2 shows five days of energy consumption by the popular Android Facebook client for participants that used the app at least once during that time, with the heaviest users over the entire study sorted to the top. While some participants display consistent consumption, others show high amounts of per-day variation. This temporal variation motivates the design requirement that Jouler enable automated energy management, since manual user reconfiguration will struggle to keep up with this level of per-app variation.

1.4 — Popular Apps Consume Most Energy

The relationship between interactive usage and energy consumption is also important when attempting to manage energy on smartphones. If app energy consumption is not correlated with interactive usage, then improvements in battery lifetime may be achieved by simply removing unused or lightly-used apps. However, if consumption is correlated with usage, then lifetime improvements will require removing apps that are used often. At best, this will significant affecting the user's experience; at worst, the user will be unwilling to remove these popular apps. Unfortunately, Figure 3 shows that energy consumption is in fact dominated by the most-used apps. As a result, as opposed to approaches such as Carat [18] that attempt to encourage users to remove apps labeled as "hogs", Jouler focuses on controlling the energy consumption of the apps users have chosen to use, not on attempting to remove top energy consumers.

1.5 — Apps Consume Energy in the Background

Given the interactive nature of smartphones, we believe that user expectations about energy consumed by apps is largely dictated by the amount of time they spend interacting with the app. However, our study revealed that on most participants' smartphones a great deal of energy is consumed by apps while running in the background—that is, while not using the display—and that patterns of app background energy usage vary significantly between devices. Figure 4 shows per-participant ratios of background and foreground energy usage during the study period. The graph shows a high variation between participants, and that many participants smartphones consume a significant fraction of their battery outside of periods of interactive use. Ideally, apps should attempt to minimize background energy consumption by adapting to usage, and Jouler is designed to help them accomplish this task (§ 2.4).

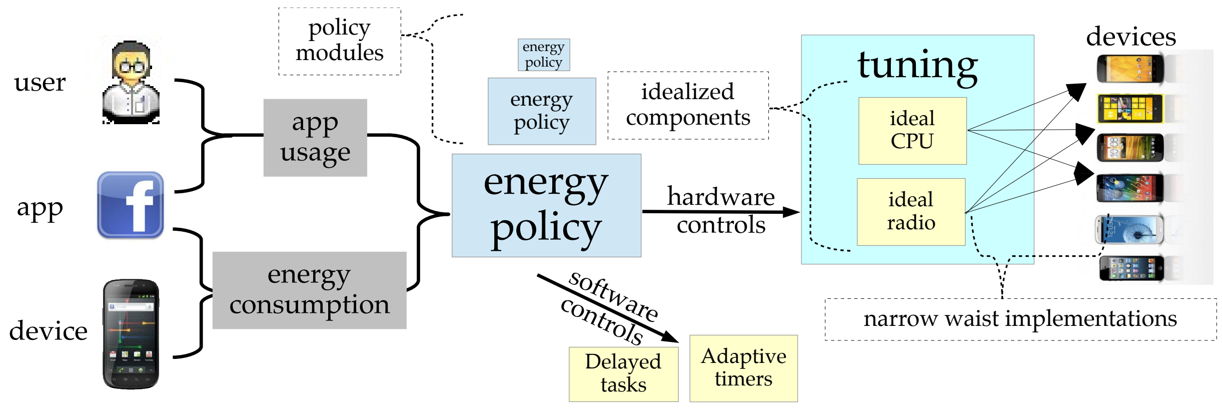


Figure 5: **The Jouler cross-device energy management framework.** Jouler addresses three key weaknesses of current approaches to smartphone energy management: the lack of additional information required to interpret app energy consumption; inflexibility in responding to inter-user, inter-app and temporal energy consumption variation; and the impossibility of deploying cross-device policies.

2 — The Jouler Energy Management Framework

The results presented in the last section motivate our goals of making Jouler *automated*—in order to respond to temporal variation—and *flexible*—in order to support the diverse requirements of smartphone users. They also indicate that in order to improve energy management we need to control existing and popular apps, since they consume most energy, but also understand the relationship between energy consumption and usage since this varies significantly between apps. Finally, in order to be *cross-device* and work across all existing and future Android devices, Jouler must be fully-integrated into the Android platform and able to operate multiple collections of hardware components.

Figure 5 provides an overview of the Jouler architecture, discussed in more detail below. Jouler embodies the classic systems design principle of separating policy—which determines how energy is allocated—from the mechanisms that measure and control energy consumption. Today's smartphone architectures rely on apps to control their own energy consumption, and both lack effective mechanisms to limit app energy consumption while also relying on apps to set their own policies. By moving policy outside each app and creating enforcement mechanisms, Jouler both puts smartphone users in control of app energy consumption while creating a more flexible policy system, since different users can apply different policies to the same app to fit their usage and battery lifetime expectations. After providing a brief overview, we discuss each component of the Jouler architecture in detail, describing what we propose to design and build and how each part of the complete system will be evaluated.

2.1 — Overview

Due to its tight integration with the Android platform, users do not install Jouler—rather, it comes included in future version of Android designed to support more flexible energy management. Jouler utilizes Android's existing measurement and modeling approaches to determine how much energy each app is consuming and the breakdown of energy consumption between components, but augments this with additional information about how the app is being used so that energy consumption can be put into context and app utility determined (§ 2.2). These two sets of information are provided to user-installed policy modules (§ 2.5) which use them to control each apps access to energy resources. Jouler will include a set of common policy modules that users can apply to apps at install time and reconfigure later if necessary, but the policy module architecture is open meaning that users can create their own policies or download ones from policy marketplaces that better suit their needs. Jouler provides policy modules with two mechanism to control app energy consumption: the ability to configure and control hardware resources for each app in the form of idealized components (§ 2.3), which allows policy modules to be portable across different devices; and the ability to interact with modified apps directly through software controls (§ 2.4), allowing energy-aware apps to use Jouler to help better tune their energy consumption.

2.2 — Determining App Utility

Smartphone energy modeling provides today's users with easy access to information about the energy consumed by the apps running on their device, a feature incorporated into all major smartphone platforms. However, consumption data alone has many significant weaknesses, highlighted by the data we have presented earlier. First, measuring consumption along does not consider inherent differences between app functionalities and usage that naturally cause differences in energy consumption. Streaming music clients, for example, should not be measured with the same yardstick as a text chat app. When they are, apps that naturally consume a lot of energy are targeted unfairly, while apps that naturally consume less have little incentive to improve their energy consumption. Second, most consumption-driven approaches try to generate global labels that fail to address differences in user behavior that can render the same app efficient on one device but wasteful on another. Finally, considering consumption alone biases interventions towards frequently-used apps, the ones that users are least likely to stop using, and ignores energy drain caused by lightly-used apps. Fundamentally, consumption information alone requires a human in the loop to provide additional context, creating additional burden on users while rendering automated approaches such as Jouler impossible.

Making energy consumption information useful to policy modules requires not only measuring how much energy an app consumed, but also providing policy module inputs allowing them to determine how much benefit or utility the app provided to the user. Thus, an apps overall *energy efficiency* can be computed as utility per unit energy and used to decide how to allocate available energy resources. Given the progress made in measuring energy consumption, measuring efficiency only requires estimating utility, although this is a challenging and open question. An effective utility metric should be easy to compute without any user involvement and applicable to a broad range of apps. It should naturally distinguish between apps that inherently consume a great deal of energy and those that do not, while enabling personalized recommendations that incorporate how efficient an app for each user. Measuring energy efficiency should easily identify apps that are consuming energy while delivering little to no utility—ideal candidates for removal or throttling that users are unlikely to miss—while providing feedback to developers interested in improving the energy consumption of their apps. Providing a second set of usage-based inputs to policy modules is what makes Jouler's automation possible.

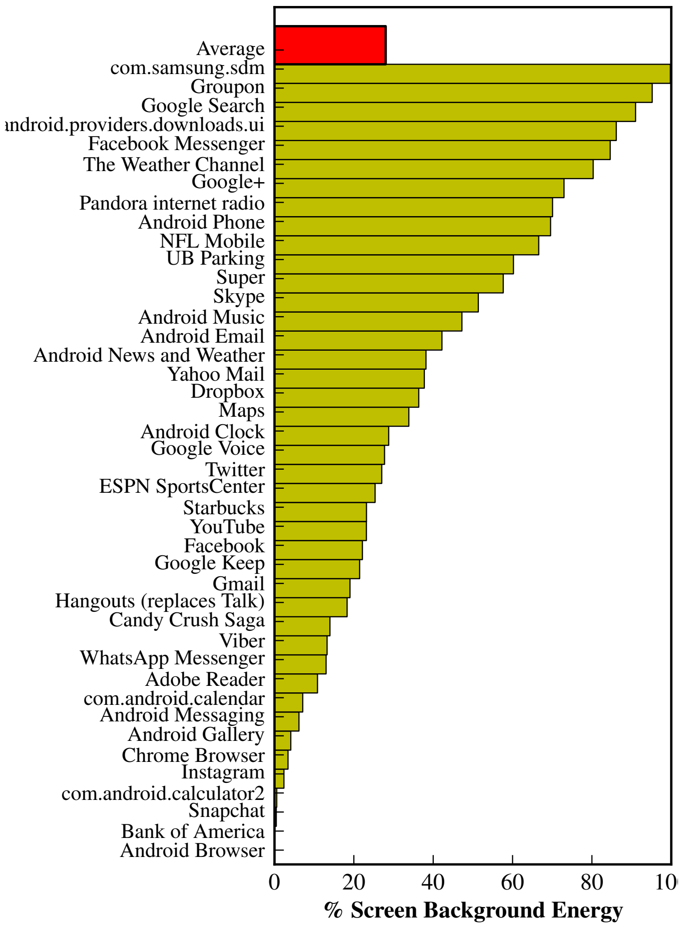


Figure 6: **Per-app background energy consumption.** A great deal of unexplained variation can be seen in per-app background energy consumption.

We propose to investigate a novel set of usage-based metrics as inputs to the Jouler policy module framework that we believe we help better understand app energy consumption and allow policy modules to do more than make decisions based on consumption rates alone. Below we summarize additional inputs Jouler will make available to policy modules alongside information about overall energy consumption and per-component energy consumption breakdowns.

2.2.1 — Foreground and background time and energy consumption: One of the simplest statistics to measure is the foreground and background time and energy consumed by each app, and some Jouler policy modules may be able to act on this information alone. As our data analysis has previously shown, a substantial amount of overall energy is used by Android apps running in the background. Figure 6 shows additional experiment data showing that a great deal of per-app variation in background energy consumption exists. One example Jouler policy module discussed in Section 2.5 simply limits the amount of energy apps can use in the background, since this use may represent poor adaptation to user behavior: for example, an app that is regularly downloading new content despite extremely infrequent usage.

2.2.2 — Screen and audio refresh rates: We believe one promising approach that may work well for many apps is to treat them as content delivery vehicles that convert energy (network traffic, processor decoding, screen redraws) to information (text, images, video). In this model, the rate at which the app is expected to consume energy becomes both a function of the amount of information the app is delivering and its efficiency in using the device. This perspective corrects one fundamental weakness of relying on foreground time as a proxy for utility, since it allows a well-written video-playing app to have a higher energy consumption rate than a chat app given that the video player is delivering more content. Other examples of information delivery include fetching a new email and presenting it to the user, as Gmail would do; or rendering the environment for a game and redrawing the screen, as Angry Birds would do.

In order to compute a measure of information Jouler monitors the usage of the smartphone's information-delivery interfaces, including the display and audio output, and provides this information as an input to policy modules. At minimum, screen and audio frame or refresh rates may provide a starting point that does not require Jouler to monitor the content flowing to these interfaces. However, we are also exploring ways to instrument Android to capture more metadata, such as the extent of the region of the screen being redrawn or the audio quality level. This will allow Jouler policy modules to distinguish between refreshes of a large region of the screen that might be caused by a video or game from redraws to a small region that could be the result of a spinner or other low-information UI element.

To fully determine information content, however, it may be necessary to provide some access to content-based statistics or the content of the video or audio stream itself. We propose to investigate how to effectively modify the Android platform to instrument common content-delivery interfaces to provide this information to Jouler policy modules. An example would be capturing text rendered by an email or chat client before it is converted to pixels, since it will be easier to determine the information content of the text itself than operate on the raw image data delivered to the screen buffer. For image and video data, it may be helpful to know something about the source to determine its quality and compressibility, since these factors influence the amount of energy the app required to obtain and render the image.

2.2.3 — User interaction statistics: Another stream of potentially-useful data for policy module energy management decision making is statistics about user interaction with each app, which could include the rate of interface events, keypresses, or other forms of interaction or data entry. We propose to investigate what kind of user interaction information policy modules can benefit from and instrument the Android platform to provide this data to Jouler policy modules.

2.2.4 — Evaluating app utility determination: One of the difficulties in investigating how to automate the process of determining app utility will be closing the loop by comparing with some other more direct measure of utility. However, the PhoneLab testbed allows us to perform survey-based experiments and we will use this capability to test the effectiveness of the usage inputs provided by Jouler in several ways.

First, once we have distributed platform changes allowing Jouler to collect the data described above we will distribute a simple app that periodically asks participants to stack-rank the importance of each app on their phone. We can then use this information directly to determine which usage inputs correlate with direct participant ranking and which do not. Second, once the Jouler framework controls are implemented we will perform an experiment where we ask a set of participants to install a simple policy module that controls one hardware component—processor frequency, for example—in response to an energy-efficiency metric computed using the mixture of the above inputs determined by the first experiment to be most effective. Half of the participants will receive this policy module, and half will receive a placebo policy module that does not do any tuning. After several weeks, we will ask each participant about the battery lifetime and responsiveness of their smartphone and compare the experiment and control groups.

2.3 — Hardware Control Through Idealized Components

The inputs to Jouler's policy modules are the energy consumption and app usage information previously described. Policy modules control app energy consumption both by adjusting hardware hardware components, described next, and through a set of software controls that allow adjusting app behavior, described in Section 2.4. These two energy control planes are complementary and allow Jouler to control app energy consumption for both modified and unmodified apps and in cooperative and non-cooperative ways. Note that in the discussion that follows we assume that Jouler has allowed an app to use the hardware resource and what is being determined is how the resource should be operated. Jouler policy modules can also deny access to hardware resources to apps entirely by preventing them from running.

Variation in hardware energy management interfaces provides the biggest challenge to enabling the Jouler system across many devices. Controlling hardware energy consumption requires controlling the hardware components that consume energy, but this is difficult to impossible when each provides a different interface to different features. For example, Wifi chipsets on smartphones provide a variety of potentially-useful energy-performance knobs, allowing the sleep interval to be altered, power levels adjusted, and encodings to be changed. Unfortunately, however, not all chipsets support all of these features, may expose them in different ways, and it can be difficult to reason about how they combine to affect meaningful performance characteristics such as latency or throughput.

To address this challenge, Jouler incorporate a novel narrow waist for implementing energy management utilizing idealized devices. An idealized device trades off energy for some declared performance characteristic in a smooth, though not necessarily linear, way. When actual components register with Jouler, they indicate which energy-performance tradeoffs they can perform. A DVFS processor, for example, would declare that it could trade off speed for energy consumption and the range over which this tradeoff can be performed. A Wifi chipset might register that it can trade off both latency and throughput separately for energy consumption. Policies in Jouler are then implemented on top of these idealized components and their common interface and use them to control energy-performance tradeoffs on a per-app basis. An app that is being allocated more energy would be allowed to operate the radio in high-bandwidth settings, while an app that is being throttled would be required to operate the radio in a lower-power state. Translation between the idealized component and the actual component is done by a component-specific piece of code that can be implemented by the device driver maintainer or anyone with specific knowledge about that piece of hardware. Thus, Jouler allows energy management policies to be written without of hardware-specific details and transferred easily between devices.

While Jouler's idealized component interface can hide some complexities of interacting with hardware components, it cannot hide all details from policy modules. For example, two processors will be able to scale frequency over different intervals and with resulting differences in power consumption. We also want to enable usage of features such as buddy-core architectures and emerging hardware capabilities such as computational sprinting [20]. Some of these we may be able to hide behind the Jouler hardware control interface, but others will need to be exposed to policy modules and may create conditional logic inside the modules themselves if they cannot be incorporate into a smooth idealized transition. Another challenge is controlling hardware needing long time intervals to change states, which may require the idealize device driver implement hysteresis to prevent overly-rapid transitions. More research is needed in this area to determine the right tradeoff between enabling cross-device policies and providing access to all hardware energy management features.

2.3.1 — Evaluating Jouler hardware control: While eventually we believe that energy control interfaces should be provided by Linux device drivers directly, as a starting point we propose to implement per-component wrappers for the processor and Wifi chipset. as part of the Android platform that conform to the Jouler hardware control API and configure hardware appropriately. Controlling processor frequency from the platform can be performed by disabling the Linux frequency governor and then writing values through the `sysfs` filesystem; power levels and sleep parameters for Wifi chipsets can be controlled similarly. While we would like to control other energy-consuming hardware components such as the 3G radio and GPS hardware, the lack of open-source drivers for these components makes this impossible.

To validate that the Jouler hardware control API can support multiple devices, we will implement these wrappers for several devices that we are able to build full platform images for, including the Samsung Nexus S 4G [1] used on PhoneLab from 2012–2013, the Samsung Galaxy Nexus [32] currently in use, and whichever smartphones are deployed on the testbed during the proposed project period. We will also develop a set of benchmarks as part of a testing suite to allow developers to validate that their implementations of the hardware controllers are operating as expected, with ground-truth power consumption data provided by an external power meter such as the Monsoon which we have available in our lab.

2.4 — Software Control Through App Interaction

Controlling hardware components directly allows Jouler's policy modules to limit the energy used by unmodified apps. However, we believe more effective app energy consumption control can be achieved by providing mechanisms to help apps manage energy while still being controlled by the policies set by Jouler policy modules. We propose to develop an interface allowing collaborative energy management between Jouler policy modules and modified apps. Two potential mechanisms we propose to explore and expose through this interface are described below: energy-adaptive timers, and energy-delayed tasks.

2.4.1 — Energy-adaptive timers: A common primitive for interactive smartphone apps is to periodically update a cache of data so that the latency when an interactive session begins is minimized. An example is an email client which periodically polls for and downloads email. Recent studies have shown that this periodic tasks can consume a great deal of energy on smartphones partly due to the "long tail" energy consumption caused by the typical operation of 3G radios [34], and so this can be provide a way for modified apps to allow Jouler policy modules to control their energy consumption. Instead of registering for a timer to fire at a fixed rate, Jouler provides *energy-adaptive timers* that fire at rates determined by the policy module assigned to control that app. When registering to receive events generated by energy-adaptive timers, apps can request a minimum and maximum timer rate to provide some degree of predictability to timer events.

Energy-adaptive timers provide a mechanism allowing apps to offload the policy decision of when to perform certain tasks directly to the policy module. As an example, consider the Facebook app updating the view of a users news feed. By using an energy-adaptive timer, Facebook can allow Jouler to determine how often to update the feed. If a user is a frequent user of the Facebook app, the policy module may allow the check to happen often, since it eliminates the delay caused by updating the feed when interactive sessions are initiated. However, if the user rarely uses or stops using the Facebook app, then the energy-adaptive timer will approach the maximum timer interval, slowing the feed updates and reducing the associated energy consumption. Energy-adaptive timers are similar to Eon's adaptive timers [24], but to our knowledge this idea has not been explored on smartphone devices.

2.4.2 — Energy-delayed tasks: One challenge to programming smartphone apps in an energy-efficient manner is understanding the variation in networking and battery conditions in order to schedule delay-tolerant tasks. For example, a backup app may need to upload a file to a server within a specific time bound, but want to do so when networking conditions will allow the transfer to be done most efficiently in order to minimize impact on the user's battery lifetime. At present, the app would have to monitor networking conditions itself and try to use this data to schedule the upload in a way specific to each user, which both wastes energy due to monitoring overhead while confining potentially-reusable logic to a single app.

Instead, Jouler provides an interface allowing apps to schedule energy-delayed tasks which are run at a time chosen by the policy module. Energy-delayed tasks are similar to energy-adaptive timers, but apps must also annotate the task request with a deadline and information about the resource that it wants to try to optimize. In the example above, the backup app would submit an energy-delayed task request indicating its backup deadline and that its task consumes energy primarily as a function of network bandwidth. The Jouler policy module would then determine when to schedule the task based on its information about each user's connectivity and charging patterns. If the policy module believes a charging event will happen, or if the user is currently connected to a mobile data network and Jouler believes that they will connect to a more energy-efficient Wifi network soon, it may delay the task until one of these events happens.

2.4.3 — Evaluating software controls: As we develop the Jouler framework we will rewrite several core Android apps with source distributed by the AOSP project to use the Jouler software controls. Similar to the approach describe in the previous evaluation, we will perform a blind study where one half of the PhoneLab participants receive platform updates with the modified apps while the second half do not receive modified apps. All participants, however, will receive the same policy module, which will control the unmodified apps entirely through hardware controls and the modified apps through a mixture of hardware and software controls. By studying the differences between the energy used by apps and participant satisfaction we can evaluate the effectiveness of collaborative energy management using the Jouler framework. In addition, during the process we expect to find other common cases where apps can offload policy decision to Jouler policy modules, and will expand the software energy management interface appropriately.

2.5 — Policy Framework

At the heart of the Jouler system is the policy module framework, which takes the energy consumption and app usage inputs described previously and uses them to control hardware and software energy consumption using the mechanisms described above. When a user installs a new app, they will be prompted as part of the installation dialog to assign the app a policy module, although this decision can be changed at any time through new functionality added by Jouler to the Android platform as a settings dialog.

The configured Jouler policy module for each app is allowed to run before and after the app begins running in either the foreground or background and at intervals during its operation. Each time the policy module runs, it is allowed to reconfigure hardware components for the app it controls, or stop it from executing entirely if the app is executing in the background. To avoid disturbing the user, Jouler will reconfigure—but not halt—apps about to enter the foreground or deliver content to the audio device. In addition, to facilitate software energy control mechanisms all policy modules are also executed periodically in order to fire energy-adaptive timers or schedule energy-delayed tasks. Policy modules have access to per-module storage, allowing them to save state required to measure temporal variations in things such as charging patterns or networking conditions.

2.5.1 — Built-in Jouler policy modules: Energy management policies may prioritize energy consumption over time, between applications, or based on any other exogenous information such as the user's current activity, location, or direct input. Below we provide examples of several standard energy management policies that we will distribute as part of Jouler.

- **Charging pattern adaptation.** Preliminary results on PHONELAB show that users display varying charging habits. This policy determines the users charging behavior and then adapts energy consumption to try and devote as much energy to improving app performance while ensuring that the smartphone will survive until it is plugged in again. The policy may also prompt periodic chargers based on their location to help them remember to recharge when power is available. While smartphones are notorious for running out of energy too soon, arriving at a plug with energy to spare is also a problem, as that energy could have been used to improve performance.
- **Gameplay optimization.** Smartphone games are a rapidly-expanding segment of the application market. This policy is suited to game apps or frequent game players and optimizes the system during gameplay by allowing the processor and memory to consume more energy while slowing energy-adaptive timers and components not utilized by the game such as storage or networking.
- **Rewarding efficient apps.** Not all apps are implemented efficiently or utilized efficiently by users. This policy uses the utility to energy consumption ratio to shift energy towards apps that are running efficiently. As described earlier, calculating utility is an open research problem, but this policy module should be able to use some set of Jouler app usage inputs to produce a reasonable estimate.
- **Limiting background consumption.** As our data shows, apps vary significantly in the amount of energy they consume in the background. In some cases, this is due to poor adaptation to user behavior. This policy module sets a target background energy consumption percentage and uses Jouler's hardware and software controls to force each app's background consumption to the target level.
- **Maximum battery life.** In certain scenarios such as when users are traveling the time to the next charging opportunity is unknown. In these situations, all applications must be run as efficiently as possible, and this can be enabled via a simple Jouler policy that only exposes hardware in its most efficient settings, runs all energy-adaptive timers at their slowest setting and delays all energy-delayed tasks as long as possible.

2.5.2 — Policy module ecosystem: While Jouler will ship with a set of pre-installed policy modules, our goal is to create a vibrant ecosystem of policy modules available to meet the great diversity present in smartphone app usage and user battery lifetime expectations. Policy modules may come along with apps which can then be suggested to the user at install time, or distributed through a "policy marketplace". Users may experiment with different policies to determine what works best for their device, or be guided to an appropriate policy through tracing techniques or usage characterization tools outside the scope of this proposal. Along with the Jouler platform changes, we will provide a simple app allowing PhoneLab participants to browse and download policy modules from an external source, which will also be used during the Jouler Energy Management Challenge described in Section 4.2.

2.5.3 — Evaluating policy modules: The goal of the policy module framework is to enable enough flexibility to respond to a variety of user needs and app requirements. Once the Jouler framework is implemented and the initial set of policy modules are in place, we will conduct additional PhoneLab experiments to determine the usage of policy modules by participants. The Jouler policy module app will include descriptions for each policy module, and we will provide an instructional video helping participants choose the module that best fits their needs. We will ask each participant to select one of the standard Jouler policy modules and use it for all of their apps for a period of several weeks, after which we will ask participants to report on their experience. During the experiment, detailed energy consumption information similar to that collected by our initial study will be gathered in order to measure the overhead of Jouler, the effect of policy modules on each app, and the overall impact on achieved battery lifetime and performance.

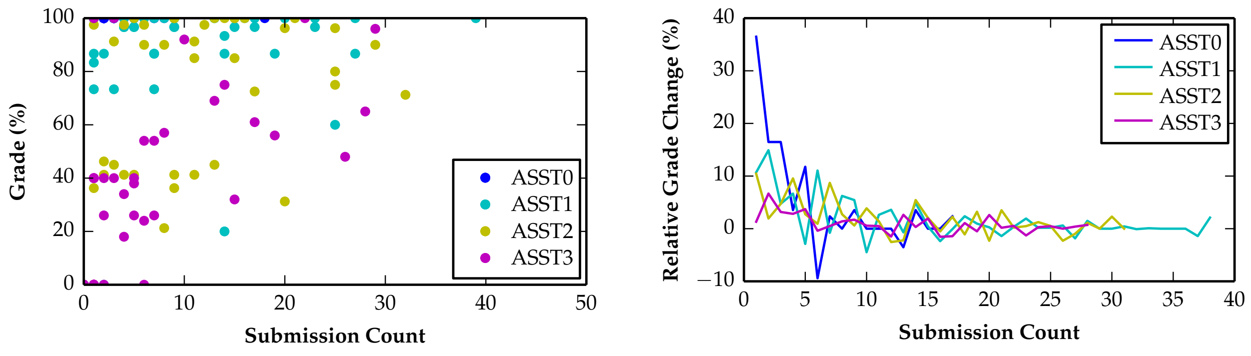
3 — Related Work

Here we briefly discuss work related to the Jouler framework, focusing on efforts to control app energy usage, either by providing users feedback on energy usage or by directly changing app behavior. 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 in cross-device manner.

Some of the most recent work on smartphone energy management has used measurements from large user communities to overcome the multiple limitations of single-user relative measurements. The Carat app has been installed by over 500.000 devices, and attempts to identify two types of energy anomalies, so-called *bugs* and *hogs* [18]. Carat defines a hog as an app that consumes higher-than average energy on multiple devices, and a bug as an app which consumes much more energy or some devices relative to others. Hogs may be due to programmer error or inherent app functionality, while bugs may be due to misconfiguration or poor interaction with certain devices or platform versions.

Carat overcomes the limitations of the energy-measurement interfaces on modern smartphones—which typically only provide access to the slowly-changing battery level—by compiling data and observing patterns across many devices. However, it cannot overcome the inherent limitations to all approaches that focus solely on energy consumption. In particular, because Carat cannot measure the amount of time that users spend interacting with apps, apps that produce higher usage and associated energy consumption are likely to be classified as hogs regardless of how well-written they are. This has the consequence of not only penalizing popular apps, but also targeting *exactly* the popular and heavily-used apps that users are most likely to remove or stop using. And while Carat's results claim that Carat users achieve a 23% improvement after two months of using the app, only 25% of users continued using the app after one month, suggesting that the improvement results may have been due to overrepresentation of users that had found the app effective and continued to use it.

More importantly, as the authors admit, Carat cannot distinguish between energy consumption caused by "coding error" from that that is "intrinsic to the app's function". We believe that this is a critical limitation of this approach that must be addressed in order to enable fair comparisons between smartphone apps. An inability to distinguish between legitimate resource needs and energy waste due to unnecessary or buggy consumption impacts both apps that legitimately require a lot of energy to function and those that do not. The former are unfairly targeted, while the latter are effectively shielded from scrutiny. Jouler's design, which provides more information than just energy consumption to policy modules, is a response to the problems with consumption-only approaches such as Carat. In addition, operating at the app level, Carat cannot control app energy usage, only report it to the user which must then take action.



(a) Relationship between grade and submission count. (b) Incremental per-submission improvements.

Figure 7: **Results from the `ops-class.org` experiment with unlimited submissions.**

3.1 — Previous Mobile Systems and Sensor Network Projects

Taking a empirical and adaptive approach to energy management of the entire network, researchers in the mobile systems community have proposed energy aware operating systems and decentralized energy management for wireless ad-hoc networks [4, 12, 14, 28, 24, 16]. Systems such as Odyssey [8], PowerScope [9] and more recently Cinder [21], have addressed measuring or adapting to energy variations on battery-powered devices, primarily to support mobile applications, while systems like Quanto [7, 10] provide new ways of profiling the power consumption of applications. Currentcy proposed a system where each process was given the an energy budget to spend and would use this currency to schedule tasks [35]. The Koala OS [23] uses runtime models to predict software power consumption and then trades performance for energy savings, but does not provide a policy framework similar to Jouler. Sensor network programming frameworks such as Eon [24] and Pixie [14] provided the ability for sensor network programs to adapt to changing energy availability, but did so by relying on special languages or program structure, approaches that would be intractable to apply to existing smartphone platforms.

4 — Broader Impacts

The broader impact components of our proposal will advance discovery and understanding by promoting teaching and learning while contribute to shared educational infrastructure.

4.1 — `ops-class.org` Online Education Module

Our excitement with computer science began with an introduction to operating systems course at Harvard taught using the OS/161 [11] instructional operating system. We brought this approach with us to the University at Buffalo as “CSE 421/521: Introduction to Operating Systems” in order to share the excitement of hands-on system design with new generations of students. Adapting the course for the University at Buffalo proved challenging due to the significant difference in course staffing levels when compared to Harvard. In response to this need, we created a new online operating systems educational called `ops-class.org`. It restructures the OS/161 assignments to allow them to be submitted through a web interface and graded almost entirely automatically, freeing the course staff to spend their time interacting directly with students through office hours and help sessions. Since CSE 421 is a work-intensive course, this extra help is appreciated by students. In addition, the automation also allows students to submit assignments repeatedly without creating any additional burden on the course staff, allowing them to incorporate feedback from the autograder and improve their submissions. Figure 7 shows data from this experiment confirming both that multiple submissions improved student outcomes (Figure 7a) and that students were able to incrementally improve their performance over multiple submissions (Figure 7b).

Our intention is to make the `ops-class.org` assignments available for free and public use, and we hope to see them completed by students online as well as the framework used to support other OS/161-based operating systems courses—COMP3231 [25] at the University of New South Wales, CSE 430 at Arizona State University [2], Computer Science 372 [6] at Denison University, and others [6, 26, 27]. Operating this educational platform at a larger scale offers many advantages, including the ability for students to compete across institutions and improved plagiarism detection due to the larger volume of stored submissions.

Courses based on `OS/161` are typically structured around three major programming assignments—(1) system calls and process support, (2) virtual memory management and (3) filesystems. We believe that omitting energy as a cross-cutting design consideration at this stage of a computer scientist's educational development misses a important chance to educate them about a critical modern design constraint. To address this, we propose adding support for a fourth major assignment on energy management, distributed through the `ops-class.org` online instructional environment. This assignment will both allow students to engage in energy-aware systems programming, while also distributing discoveries from the research objectives of this project to a wide audience.

4.2 — Jouler Energy Management Challenge

To help build awareness of the availability of the Jouler platform and competency in writing policy module, we will hold a public Jouler Energy Management Challenge during the last year of the proposed project. By that point the main components of Jouler will be developed and deployed on PhoneLab allowing us to distribute policy modules to participants.

The competition will work as follows. Entrants will receive access to the source code of a simple sample application and be asked to write a Jouler policy module that, when attached to that application, minimizes its overall energy usage across all experiment participants subject to a set of functional constraints. PhoneLab participants that agree to participate in the experiment will install the app and one of the policy modules provided by the challenge entrants. In order to reach a representative subset of PhoneLab participants, the entrant count will either be limited or a series of week-long experiments will be run, allowing each policy module entered to run for a period of time on a large enough sample of PhoneLab participants to eliminate inter-participant usage variations. The results from the competition will be submitted to an appropriate conference or workshop with the entrants as co-authors.

4.3 — Curriculum Development Activities

Development of the Jouler framework will occur in conjunction with a research seminar taught by PI Challen. The graduate course will focus on the design and implementation of Jouler, and provide interested students with a unique opportunity to develop within the Android platform and distribute updates using the PhoneLab testbed. This seminar will be taught in all three project years, with development of the Jouler framework, policy modules, idealized hardware wrappers and other components occurring sequentially in each year as described in the preliminary task plan (§ 6.2). In the final year of the project, seminar students will help organize the Jouler Energy Management Challenge described above, as well as participating through their own policy module entries.

4.4 — Diversity and Outreach

When admitting graduate students, the PI is inspired by the approach of the CCC/CRA NSF-funded CI Fellows program, which improved representation of women and traditionally-underrepresented minorities increased simply by reviewing those applications first. The PI is copying this approach when admitting graduate students and recruiting undergraduates. Over the past several years, five women have contributed to our research groups: Rizwana Begum (Drexel), collaborator; Vinu Charanya and Anuja Raval (University at Buffalo), Masters students; Sonali Batra, Na Gong, and Anudipa Maiti (University at Buffalo), PhD students. We are expanding our efforts to recruit females and underrepresented minorities earlier in their careers. With its large population of undergraduates from underrepresented groups, the University at Buffalo is an excellent place to cultivate future researchers.

As another creative way of recognizing and encouraging diversity within computer science, we are leading the commissioning of a "Diversity in Computer Science" mural that will be installed on a large wall in the Computer Science and Engineering Department at the University at Buffalo. Students will be encouraged to submit their images celebrating diversity with a prize and mural installation funded by contributions from faculty. PI Challen initiated and is coordinating the mural project.

4.4.1 — Undergraduate Research: We are committed to involving undergraduate students in research. In the past year, PI Challen has involved two talented undergraduates in PhoneLab projects. Frank Rossi assisted in the development of the PocketLocker system, which creates personal storage clouds from available storage on multiple personal devices, while Nick Dirienzo is working on a system called PocketMocker that allows smartphone users to conceal their true activities from applications by injecting fake data. As a general philosophy, we integrate undergraduates into our research team by inviting them to work alongside graduate students in the lab and giving them development tasks appropriate to their training.

5 — Qualifications, Deliverables and Plan

In this section we describe why we are qualified to undertake this project, list project deliverables, and present a plan outlining how we will complete the required tasks within three years.

6 — Qualifications and Prior Support

PI Challen is a young investigator that brings his previous experience with energy management, hardware-software co-design, utility measurement, sensor networks, and online education to the project. His previous work on Lance, which used policy module to enable utility-driven wireless sensor network bulk data collection [29], provides an ideal starting point for Jouler. Jouler also must estimate app utility in order to put energy usage into context, and also utilizes policy modules to achieve flexibility within a well-defined framework. He also developed energy management middleware called IDEA for wireless sensor networks which performed local automatic parameter to maximize objectives defined across the entire network.

In addition to Lance and IDEA, he assisted on two other energy management framework projects. The Pixie [15] sensor node operating system promoted energy to a first-class resource, requiring programmers consider energy when developing sensor network applications. Peloton [28] proposed a distributed operating system for coordinated resource management built on state sharing, a distributed energy ticket abstraction, and neighborhood ticket management. PI Challen also operated MoteLab [31], a large-scale programmable sensor network testbed that was actively used by the sensor network community; assisted in the development of PowerTOSSIM [22], an augmented version of TOSSIM [13] enabling application power profiling; and deployed wireless sensor networks on active volcanos [30].

Since beginning his faculty appointment at the University at Buffalo PI Challen has focused on energy management in mobile systems and leading the effort to build PhoneLab, the world's largest programmable smartphone testbed. His group has designed and deployed software to operate and manage the testbed, a toolkit for processing and analyzing collected data, and deployed a usage characterization experiment collecting a variety of useful data during a beta testing year prior to beginning active experimentation. During the last year, PhoneLab has added the ability to distribute platform updates, facilitating modifications to core Android platform components and the Linux kernel that cannot be distributed at scale in any other way. His group is currently using this capability to study several low-level aspects of smartphone behavior, including the energy usage experiment described earlier and an experiment exploring file access patterns. His work on energy management has coined the term "power agile computing" as part of an ongoing effort to redesign interfaces between the Android platform, operating system, and hardware to better facilitate adaptive energy management on hardware with emerging energy-proportional features.

In addition to his work on energy management, PI Challen is leading efforts on improving smartphone sustainability through discarded device reuse [3], helping smartphone users improve their security through objective-based context mocking, using smartphones to prepare for and survive disasters, interaction-free crowdsourcing or "pocketsourcing", and new approaches to combining available storage on multiple devices into reliable personal storage clouds. PI Challen is also the creator and developer of `ops-class.org`, an online operating systems instructional platform integrating tools for human- and computer-driven grading, video and slide delivery, as well as plagiarism detection. Already in use at the University at Buffalo, `ops-class.org` provides both a source of actionable information and a laboratory for testing new data-driven approaches to teaching and learning, and will serve as the delivery platform for the proposed new energy management assignment.

His current NSF support is from **PhoneLab: A Programmable Participatory Smartphone Testbed** *(CI-ADDO-NEW-1205656, $1.3M, 06/01/2012–05/31/2015)*, which he leads as PI. PHONELAB's accomplishments include:

- **Intellectual Merit Accomplishments:** PHONELAB opened in October, 2013, and has already received six external experiment requests and produced one publication [17].
- **Broader Impacts Accomplishments:** PHONELAB software will be made available to researchers interested in running their own testbeds during the coming year.

6.1 — Deliverables

Our primary project deliverable will be the implementation of the Jouler platform, representing a set of changes to the Android platform. The sources for the Jouler version of Android will be made publicly available through share source code repositories to facilitate verification and validation of our results and further projects that attempt to continue the proposed research. We will also submit patches with the Jouler changes to AOSP maintainers in hopes of having portions of the framework reach mainline Android platform distributions. We will also approach the CyanogenMod community about including our changes in their popular modified distribution, which is maintained separately and generally considered more experimental than the AOSP.

6.2 — Preliminary Task Plan

Below we describe how we will complete the development of Jouler and related broader impact activities within three years.

6.2.1 — Year 1: In the first year we will begin development of the core Jouler components, including implementing idealized component wrappers for the Galaxy Nexus processor and Wifi chipset and augmenting the platform to collect app usage information. One graduate student research assistant will be assigned each of these tasks. With this new input and controls, we can begin building Jouler policy modules, and the goal for the year will be to complete the development of the first policy module and deployment on smartphones used by PHONELAB developers.

Some of the development will occur as part of the Jouler seminar, which will begin in the first year. PI Challen will also begin writing the new energy management assignment to distribute on the `ops-class.org` online operating system educational framework, with the goal of using it as part of his introductory operating systems course during the second year.

6.2.2 — Year 2: During the second year the goal will be to complete the evaluation of the hardware controls and app usage information while developing the software controls through changes to the Android platform and writing new policy modules. Each graduate student will be responsible for evaluating their first-year contribution to Jouler during the second year, and one will be assigned to complete the software control development while the second works on implementing additional policy modules. The goal during the second year will be to complete the development of the full framework and deploy it on all PHONELAB participant smartphones through a platform update, which will be necessary to perform a complete evaluation of the full system during the third year.

The graduate research seminar will continue during the second year and students will be engaged in helping test Jouler, implement new policy modules, and modify existing Android apps to use the novel software energy management controls provided by Jouler. In addition, the new `ops-class.org` energy management assignment will be used by PI Challen as part of his introductory operating systems course before public release in the third year.

6.2.3 — Year 3: The last project year will focus on evaluating the Jouler framework through extensive deployment on PHONELAB while also publicizing the changes and pushing for them to be adopted by Android platform maintainers. Both graduate students will be involved in analyzing data from the PHONELAB deployment of Jouler begun in the second year and correcting any problems that this data reveals.

During the final year the Jouler Energy Management Challenge will take place, organized by the two graduate research assistants and with assistance from the Jouler seminar which will take place for the third time. Finally, the `ops-class.org` energy management assignment will debut for free and open use.

REFERENCES

[1] Nexus S 4G from Google. `http://www.android.com/devices/detail/nexus-s-4g-from-google`.

[2] Arizona State University. CSE 430 - Operating Systems - Fall 2010. `http://www.eas.asu.edu/~cse430/`.

[3] G. Challen, S. Haseley, A. Maiti, A. Nandugudi, G. Prasad, M. Puri, and J. Wang. The Mote is Dead. Long Live the Discarded Smartphone! In *Proc. of the 15th Workshop on Mobile Systems and Applications (ACM HotMobile 2014)*, February 2014.

[4] G. Challen, J. Waterman, and M. Welsh. IDEA: Integrated Distributed Energy Management for Wireless Sensor Networks. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010)*, June 2010.

[5] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. C. Hu, and A. Rice. 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*, SIGMETRICS '13, pages 29–40, New York, NY, USA, 2013. ACM.

[6] Dixie State College of Utah. CS 3400 Operating Systems. `http://cit.dixie.edu/cs/cs3400/asst0.php`.

[7] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *International Conference on Information Processing in Sensor Networks (IPSN'08)*, April 2008.

[8] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. *SIGOPS Oper. Syst. Rev.*, 33(5):48–63, 1999.

[9] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, page 2, Washington, DC, USA, 1999. IEEE Computer Society.

[10] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, November 2008.

[11] D. A. Holland, A. T. Lim, and M. I. Seltzer. A new instructional operating system. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, SIGCSE '02, pages 111–115, New York, NY, USA, 2002. ACM.

[12] A. Lachenmann, P. J. Marron, D. Minder, and K. Rothermer. Meeting lifetime goals with energy levels. In *ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, November 2007.

[13] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proc. the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 2003.

[14] K. Lorincz, B. rong Chen, J. Waterman, G. Werner-Allen, and M. Welsh. Resource Aware Programming in the Pixie OS. In *ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, November 2008.

[15] K. Lorincz, B. rong Chen, J. Waterman, G. Werner-Allen, and M. Welsh. Resource aware programming in the pixie os. In *Proc. of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, pages 211–224, New York, NY, USA, 2008. ACM.

[16] G. Mainland, D. C. Parkes, and M. Welsh. Decentralized, adaptive resource allocation for sensor networks. In *Symposium on Networked Systems (NSDI'05)*, May 2005.

[17] A. Nandugudi, A. Maiti, T. Ki, F. Bulut, M. Demirbas, T. Kosar, C. Qiao, S. Y. Ko, and G. Challen. Phonelab: A large programmable smartphone testbed. In *Proc. of the 1st International Workshop on Sensing and Big Data Mining (SenseMine 2013)*, November 2013.

[18] A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma. Carat: collaborative energy diagnosis for mobile devices. In C. Petrioli, L. P. Cox, and K. Whitehouse, editors, *SenSys*, page 10. ACM, 2013.

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

[20] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. Wenisch, and M. Martin. Computational sprinting. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, pages 1–12, 2012.

[21] S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Apprehending joule thieves with cinder. In *MobiHeld '09: Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, pages 49–54, New York, NY, USA, 2009. ACM.

[22] V. Shnayder, M. Hempstead, B. rong Chen, G. Werner-Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proc. of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, November 2004.

[23] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: a platform for os-level power management. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 289–302, New York, NY, USA, 2009. ACM.

[24] J. Sorber, A. Kostadinov, M. Brennan, M. Garber, M. Corner, and E. D. Berger. Eon: A Language and Runtime System for Perpetual Systems. In *ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, November 2007.

[25] University of New South Wales. COMP3231/9201/3891/9283 Operating Systems 2010/S1. `http://cgi.cse.unsw.edu.au/~cs3231/`.

[26] University of Toronto. CSC369H: StG Info. `http://www.cdf.toronto.edu/~csc369h/winter/stg/index.shtml`.

[27] University of Waterloo. CS 350 - Operating Systems. `http://www.student.cs.uwaterloo.ca/~cs350/F10/`.

[28] J. Waterman, G. W. Challen, and M. Welsh. Peloton: Coordinated resource management for sensor networks. In *Proc. of the 12th Workshop on Hot Topics in Operating Systems (HotOS-XII)*, May 2009.

[29] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh. Lance: Optimizing high-resolution signal collection in wireless sensor networks. In *Proc. of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, Raleigh, NC, USA, November 2008.

[30] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proc. of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006)*, Seattle, WA, November 2006.

[31] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A Wireless Sensor Network Testbed. In *Proc. of the 4th International Conference on Information Processing in Sensor Networks (IPSN 2005)*, April 2005.

[32] Wikipedia. Galaxy Nexus. `http://en.wikipedia.org/wiki/Galaxy_Nexus`.

[33] F. Xu, Y. Liu, Q. Li, and Y. Zhang. 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*, nsdi'13, pages 43–56, Berkeley, CA, USA, 2013. USENIX Association.

[34] F. Xu, Y. Liu, T. Moscibroda, R. Chandra, L. Jin, Y. Zhang, and Q. Li. Optimizing background email sync on smartphones. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, MobiSys '13, pages 55–68, New York, NY, USA, 2013. ACM.

[35] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Currentcy: a unifying abstraction for expressing energy management policies. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 4–4, Berkeley, CA, USA, 2003. USENIX Association.

[36] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. 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*, CODES/ISSS '10, pages 105–114, New York, NY, USA, 2010. ACM.