Future of Mobile Software for Smartphones and Drones: Energy and Performance

Abhijeet Banerjee and Abhik Roychoudhury
National University of Singapore
{abhijeet,abhik}@comp.nus.edu.sg

ABSTRACT

The need for performance and energy efficiency in mobile devices is apparent with the obvious shifting of more intensive computation to mobile platforms. In this paper, we first make a clear distinction between performance and energy issues. Apart from showing that performance efficiency is neither co-related with energy-efficiency nor inefficiency, we focus on programming methodologies and software validation approaches for producing energy efficient mobile software. These include reviewing recent works on energy-aware programming and non-functional testing to expose energy and performance issues in mobile software. As mobile platforms continue to evolve, new scenarios and use-cases involving mobile devices are on the rise. We speculate on scenarios involving energy hungry mobile software in near future, and how existing software engineering techniques can evolve to combat energy inefficiency in such scenarios. These include the need to effectively manage the energy-consumption of software-controlled personal drones which are likely to become main-stream in near future. We suggest integration of concepts from price theory in Economics to build a distributed energy management framework for software-controlled personal drones.

1. INTRODUCTION

The mobile device ecosystem is evolving rapidly. Mobile device usage has evolved from smart-phones to tablets, smart watches and personal drones. From the time when the first Android based mobile devices were launched in 2008, to the time of this writing, nearly 12 versions of Android operating systems (OS) have been released [1]. In comparison, the biggest Desktop OS market share holder, Microsoft, has only released 4 versions of Windows OS during the same time period [2, 3]. Even the hardware powering these mobile devices has been evolving steadily. Processing power, screen sizes, resolution, sensors, cellular radios, *etc* have all evolved steadily to keep pace with the changing needs of mobile users [4]. For instance, processor designers, such as ARM, have introduced heterogeneous computing platforms such as big.LITTLE [5] architecture to address the need for better performance and energy-efficiency in mobile devices. For location sensing, mobile devices often carry an A-GPS, instead of the conventional GPS, in order to increase precision, performance and energy-efficiency. These are but a few examples which show the efforts that hardware manufactures are putting in to make devices that are more suitable for the mobile ecosystem. Unfortunately, the same cannot be said for app development side of the mobile device ecosystem. This is the topic we address in this paper.

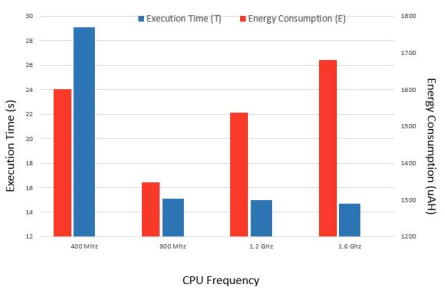
The lack of appropriate software development practices in various stages of the app development life-cycle may lead to inefficient or buggy code. This is even more so true in the case of non-functional properties, such as performance and energy-efficiency. A study we conducted on $170,000$ user reviews collected from Google Play app store revealed that users are more likely to uninstall an app if it shows energy-inefficient behaviour as compared to other kinds of inefficient behaviors (*see* Section 5.2). Existing research works [6, 7] have also presented studies that show inefficient performance, energy-consumption behaviour in popular mobile apps. At this point, we would like to highlight that energy-efficiency and better performance are *not* synonymous in the context of mobile apps (*see* Section 2). In particular, there is a more urgent need to develop suitable methodologies for energy-aware, app development because of the battery-constrained nature of mobile devices. Despite the recent research in this area (*see* Section 3), developing an energy-aware programming framework is a challenge. On this topic, we shall discuss ideas that can help achieve the goal of energy-aware programming for all stages of the app development life-cycle − from design to maintenance (*see* Sections 4 and 5). In particular, we propose an automated energy-aware maintenance framework that monitors user reviews to detect signs of undesirable behaviour (such as bugs), converts them into test cases using techniques from natural language processing and subsequently helps the developer by localizing, visualizing and patching the undesirable behaviour.

Finally, we discuss a potential evolution of the mobile device ecosystem that includes personal drones. Drones, like smartphones, often face real-time constraints due to the nature of their operations. In addition, these constraints must be met while using the limited on-board battery. As a result, judicious use of on-board battery power is crucial for effective operation of drones. Developing a holistic energy management strategy for drones is challenging because it often operates in an unpredictable physical environment, as a result all the performance, energy-consumptions scenarios may not be known beforehand. To address this challenge we propose a distributed energy management strategy that uses Price Theory from Economics to seamlessly integrate several objectives, such as task-priorities, robustness, fairness and power-saving techniques (*see* Section 6).

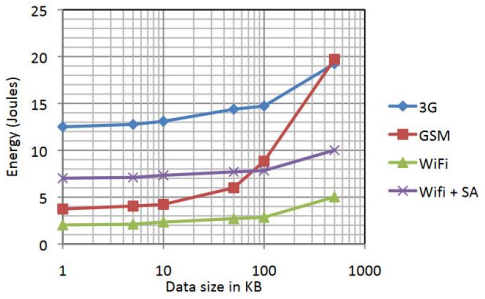
2. PERFORMANCE VS. ENERGY

In the following, we discuss the co-relation between energy-efficiency and performance in the context of mobile-devices.

CPU Scaling. CPU frequency scaling has a significant impact on performance, fluidity (of user interface) and battery-consumption of a mobile device. As a result, Android enthusiasts may choose from hundreds of automated CPU frequency scaling mechanisms (called CPU governors), that best suit their specific needs [9]. For instance, the *OnDemand* governor increases the CPU frequency as



(a) Execution time, energy consumption with frequency scaling



(b) WiFi versus 3G versus GSM measurements [8]

Figure 1: Energy-consumption measurement on Samsung S4

soon as the load hits a predefined threshold, as a result, provides good UI fluidity. Whereas other governors such as *Performance* and *Powersave* set the CPU frequency to highest and lowest, in order to achieve better performance and energy-efficiency, respectively.

```
do{
    scrape_puzzle();  // I/O intensive
    process_puzzle(); // CPU intensive
}while(more puzzles to scrape);
```

Figure 2: Relevant portions of Shortyz app

To observe the impact of CPU scaling on performance and energy-consumption we devised a simple experiment using a portion of the Shortyz crossword mobile app [10]. The portion relevant to our experiment has a structure as shown in Figure 2 and consists of scraping and processing of crossword puzzles. The scraping part of the app is I/O intensive whereas the processing part of the app is CPU intensive. We ran the app on several different CPU frequencies on a rooted Samsung S4 device, the results for which are shown in Figure 1(a). We observed that the app ran fastest when the CPU frequency was set to 1.6 GHz, while on average it took the longest amount of time when the CPU frequency was set to 400 MHz. This is expected behavior as setting higher CPU frequencies would lead to faster processing of the CPU intensive parts. However, the interesting observation is that the average energy-consumption of the app is not the smallest in either of the cases where CPU frequency is set to highest (1.6 Ghz) or lowest (400 Mhz). Instead the case where the CPU frequency is set to 800 Mhz has the lowest energy-consumption. For this particular experiment, increasing the frequency from 400 Mhz to 800 Mhz has a definite advantage as execution time drops by half, however subsequent frequency increases by the same amount (400 MHz) do not result in similar reduction in execution time. The energy-consumption of the CPU is directly influenced by the CPU frequency; higher CPU frequency causes more power consumption. Considering these observations, one might argue that energy-efficiency follows the law of diminishing returns, which states that "*adding more of one factor of production, while holding all others constant ('ceteris paribus'), will at some point yield lower incremental per-unit returns*" [11]. In this example, "factor of production" is battery-power which is used to obtain better performance, while "per-unit returns" corresponds to energy-savings as a result of shorter execution times.

Network Operations. : Network interfaces on mobile device use different communication protocols and have varying performance and energy-consumption characteristics. For instance, real-life transmission rates of cellular radio (GSM, 3G and 4G) can vary from 10 Kbps to 10 Mbps [12], whereas it is not unheard of to obtain WiFi transmission rates of a few hundred mega bytes per seconds [13]. Obviously faster transmission rates would lead to faster downloads (or uploads), leading to faster, more response apps. However, faster transmission rates may not necessarily mean lower energy-consumption for network operations. In Figure 1(b), we recapitulate an experiment reported in Section 3.6 of [8]. It simply plots the energy-consumption of various network components for transmitting data packets of different sizes. This graph clearly shows that faster is not always better, for instance 3G consumes more energy than GSM for smaller data sizes.

3. RELATED WORKS

The hardware components that constitute a mobile device, play a crucial role in determining its energy-consumption characteristics. However, actual energy-consumption of the device depends not only on its hardware components, but also on the design of the app that runs on it. Therefore, in order to develop energy-efficient apps it is important to understand both, the hardware, and the software aspects of energy-consumption. In Section 3.1, we shall discuss some of the efforts that were made to understand the energy-consumptions properties of hardware components. Next, in Section 3.2, we shall discuss some works that provide insights into the energy-consumption properties of software.

3.1 Hardware Modeling

Early works [14, 15] on hardware modelling for energy consumption were primarily focused on the CPU, possibly because CPU was the most energy-consuming component in the systems studied in those times. For instance, [14] proposes an approach for energy-aware, instruction-level energy-consumption modelling framework that could be applied to embedded system. Given a per-instruction energy-consumption model for a system, it calculates the net energy-consumption of an application (for that system) by simply adding up the energy-consumption cost of each instruction in the application. Since inter-instruction effects are not accounted for in the approach of [14], it would not be applicable to contemporary systems. More recent works [16–19] have proposed similar approaches for hardware modelling, albeit they consider more complex microprocessors in their study. They also employ a higher level of hardware abstraction when constructing the energy-consumption models. For instance, [16] proposes an energy-consumption modeling for functional blocks, such as digital signal processors (DSPs), instead of the per-instruction modelling as proposed in [14].

The imprecise results generated by some of the approaches discussed in the previous paragraph, as a result of neglecting inter-instruction effects, can be avoided by the use of cycle-accurate power simulators. SimplePower [20] and Wattch [21] are two such

cycle-accurate power estimation tools designed for the SimpleScalar architecture [22]. The simulation method itself is relatively straightforward; for each clock cycle, the tool simulates the execution of all active instructions and estimates the power consumption for each active functional unit in that cycle. Simulation continues until the halt instruction is fetched, after which any remaining instructions in the pipeline are executed before concluding the simulation. Overall, developing cycle accurate simulators is tedious, and given the fragmentation of possibilities in the mobile platform market, it is burdensome to develop cycle accurate simulators for individual processors.

3.2 Profiling, Test Generation & Optimization

In recent times, due to prevalence of smartphones and mobile apps, studying energy-efficiency of apps on such platforms has garnered substantial interest. Recent works such as [7, 23] have presented profiling-based approaches to study energy-consumption characteristics of mobile apps. In particular, [7] has shown the existence of energy-inefficient behavior in popular mobile apps such as Facebook. Another recent work [24] has extended the concept to energy-aware profiling to source-line power measurement for mobile apps. Such a tool could be of use to app developers while trying to understand the energy-impact of a given line, function or thread within their app code.

A key shortcoming of profiling-based techniques is that they require test-inputs to generate the profile. Randomly finding test inputs that lead to energy-inefficient behavior is usually non-trivial for most real-life apps. Designing a framework that can automatically generate energy-inefficient inputs for an app requires solving two challenges: (i) designing a mechanism to detect energy-inefficient behavior and (ii) designing a mechanism to systematically explore an app. It is worthwhile to know that designing a mechanism to detect energy-inefficient behavior solely based on energy-consumption data would not be possible. This is because high energy consumption by itself may not indicate energy-inefficient behavior. For instance, consider a scenario where two apps say A_1 and A_2 have similar energy consumption characteristics. However, app A_1 has a much higher utilization of its hardware resources (such as CPU, WiFi, *etc*) as compared to app A_2. In such a scenario, app A_1 is more energy-efficient than app A_2. Therefore, in order to detect energy-inefficient behavior it is important to define an appropriate metric for energy-inefficiency. The work of [25] addresses both these challenges, that are, defining a metric for energy-inefficiency and designing a framework for systematic app exploration.

There are several approaches to achieve energy-efficient software. For instance, approaches described in the previous paragraph explicitly finds test cases that reveal inefficient behavior prior to repair and/or optimization. There are however a number of works [26–28] which follow an alternative approach. For instance, the works of [26, 27] use insights gathered from empirical studies to recognize behaviors that are in general energy-inefficient and subsequently developed tools that could rectify such inefficient behavior at runtime. [26] in particular reports that small HTTP requests are common characteristics of mobile apps and can lead to energy-inefficiency. Therefore, it proposes a mechanism that can detect and bundle small HTTP requests into bigger bundles thereby reducing the overall energy consumption. The work of [27] focuses on smartphone displays. Smartphone display are one of the most power consuming components in a smartphone. Increasingly, newer smartphones are being shipped with organic light-emitting diode (OLED) displays which, unlike conventional liquid crystal displays, have energy-consumption characteristics that are heavily influenced by the colors that are on the display. In general, displaying brighter colors (such as white), on a OLED display may lead to higher power consumption. This can be a cause of concern for the smartphone battery life when browsing websites, which are often brightly colored [27]. To address this issue [27] proposes a framework that can be used to dynamically change the colors of web content while browsing on a smartphone. The work of [28] studies the use of genetic programming in reducing the energy-consumption of applications. In particular, it endeavors to generate energy-efficient versions of MiniSAT, a popular Boolean satisfiability solver.

4. POWER MEASUREMENTS

Writing energy-efficient applications for mobile platforms requires a thorough understanding of the energy-consumption characteristics of the hardware platform on which the application is intended to run. In addition, energy-aware test-generation frameworks (such as [25]), require an online power measurement technique. The work of [29] presents such a technique where the energy-consumption characteristics of Openmoko Neo Freerunner, HTC Dream and Nexus One mobile devices are studied. These devices are relatively old (launched during 2008 to 2010). Therefore, we conducted experiments to observe the energy-consumption characteristics of a few contemporary mobile devices (*see* Table 1).

4.1 Online Power Measurement

In order to fully understand the energy-consumption characteristics of an app, it must be executed on the target device. One of the earlier works [29] proposed an online power measurement technique, where instantaneous currents and voltages were measured across individual hardware components of the mobile device. Although such a technique is likely to produce precise power consumption estimates, it would be quite difficult to recreate the setup (of [29]) for more recent mobile devices. This is because unlike the highly-customizable, open-source device (Openmoko Neo Freerunner) used in the work of [29], commodity mobile devices are not very amenable to customization. Subsequent works, such as [25], therefore use a different, more abstract approach for power measurement (setup shown in Figure 3). Instead of measuring current/voltage for individual components of the device, instantaneous power consumption across the battery was measured in this setup. This meant that as long as the battery compartment of the mobile device was accessible (such as in Samsung S4), a power meter could easily be used for online power measurement. However, with the newer models of mobile devices even this method of power measurement has become obsolete. This is because current generation of mobile devices, such as those shown in Table 1, do not provide an easy access to the battery compartment. Therefore, yet another technique for power measurement is needed. Ideally, such a power measurement technique should have the following characteristics: (a) capable of online power measurement, (b) does not require access to the battery compartment, or any of the hardware components within the mobile device, and (c) does not require *rooting* the mobile device, as in some cases this may lead to voiding the warranty of the device [30].

Filtering by these criteria, we noted two approaches for online power measurement in the existing literature. They are *(i)* PowerTutor [31] and *(ii)* Dumpsys [32]. In particular, PowerTutor is a research prototype that uses its in-built power model to generate power-consumption estimates for an app under execution. To use PowerTutor, a user must install (and run) the app on the target device manually. The Dumpsys utility, on the other hand comes standard with the Android distribution and can generate battery usage information, among several other statistical data relevant to an

Table 1: Devices

Device Name 5	SoC	CPU (GB)	RAM	WiFi	GPS	Display	Battery (mAh)	Android Version
Samsung S4	Exynos 5410 Octa	Cortex-A15,A7	2	Wi-Fi 802.11	A-GPS, GLONASS	AMOLED	2600	5.0.1
Samsung S6	Exynos 7420 Octa	Cortex-A57,53	3	Wi-Fi 802.11	A-GPS, GLONASS BDS	AMOLED	2550	6.0.1
Samsung Galaxy J7	Snapdragon 617 Exynos 7870 Octa	Cortex-A57,53	2	Wi-Fi 802.11	A-GPS, GLONASS BDS	AMOLED	3300	6.0.1
Samsung Galaxy Note 5	Exynos 7420 Octa	Cortex-A57,53	4	Wi-Fi 802.11	A-GPS, GLONASS BDS	AMOLED	3000	6.0.1
HTC One M8	Snapdragon 801	Krait 400	2	Wi-Fi 802.11	A-GPS, GLONASS	LCD3	2600	6.0.0

app. In the following paragraphs we discuss a set of experiments that we conducted to observe the efficacy of these alternative power measurement techniques. The subject device for our experiments was a Samsung S4, with an Android OS v5.0.1 (see Table 1). In addition, a Monsoon power meter was used to measure the power consumption of the mobile device.

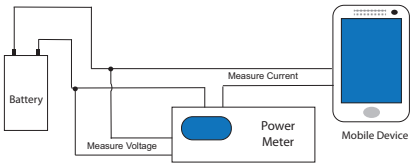


Figure 3: Block diagram for measurement setup [33]

CPU/Memory Micro-benchmark. To measure power consumption of CPU and Memory subsystems, we created a compute-intensive micro-benchmark. In particular, this micro-benchmark encrypts a string object using Advanced Encryption Standard (AES) encryption that is implemented in the $javax.crypto.Cipher$ library. To achieve a substantial runtime for the benchmark, 300 iterations of the encryption are executed. The encrypted string generated from one iteration is used as an input for the next iteration. This prevents speed-ups due to spatial-locality. Additionally, to prevent memory overflows, the length of the input string is restricted to $100,000$ characters. The micro-benchmark starts execution after the display timeout period of 15 seconds. This helps in minimizing the impact of display power consumption. Once started, the micro-benchmark does not perform any input/output operation, except for one single beep tone that indicates the completion of the micro-benchmark (after which power-measurements for the micro-benchmark are recorded). The energy measurements for the CPU micro-benchmark are shown in Figure 4.

Display Micro-benchmark. The usual behavior for mobile devices, such as smart-phone and tablets, is to go to an idle (or sleep) state, after a period of user-inactivity. In order to measure the power consumption of device display, this micro-benchmark uses the $WindowManager$ [34] utility in Android, to delay the transition of device display to an idle state. In particular, the screen is kept on for a duration of 180 seconds, while the energy-consumption is measured. The energy-consumption measurement for the display micro-benchmark are shown in Figure 4.

WiFi/SD-Card Micro-benchmark. To measure power consumption of WiFi and SD Card, we created a micro-benchmark that downloads a 100 MB file over the Internet and subsequently writes this downloaded file to the SD Card. This micro-benchmark is implemented using the $FileUtils$ API from *org.apache.commons* library. To minimize the impact of display power consumption on the measurements, this micro-benchmark starts execution after the display timeout period of 15 seconds. The energy-consumption for this micro-benchmark is shown in Figure 4.

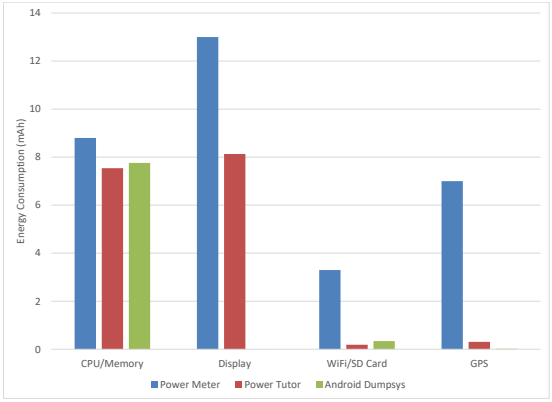


Figure 4: Power measurements for Micro-benchmarks

GPS Micro-benchmark. The GPS micro-benchmark is implemented using a $LocationListener$ [35] service, that requests for location updates on the mobile device. It is worthwhile to know that Android apps can use several strategies to acquire location of the device, not all of which may necessarily use the GPS sensor [36]. Therefore, prior to running the GPS micro-benchmark we disabled all location sensing strategies other than that provided by the GPS sensor. The location updates in this micro-benchmark were requested for a period of 180 seconds. Similar to previous experiments this micro-benchmark starts execution after the display has timed out so as to minimize the impact of display power consumption. The energy-consumption measurements for the GPS micro-benchmark are shown in Figure 4.

4.2 Observations

In Figure 4, we observed that measurements by PowerTutor and by Android Dumpsys were lower as compared to the measurements obtained by the power meter. Both, PowerTutor and dumpsys produce power consumption estimates of an app by monitoring usage of device's hardware components. The PowerTutor app also tracks the power states in which the hardware components are, while they are being used by an app. It then combines this information with its inbuilt power model to generate power consumption estimates. The research article behind PowerTutor [31] states that the power model used in the PowerTutor was built using HTC G1, HTC G2 and Nexus One mobile devices. However, in our experiments (Sec-

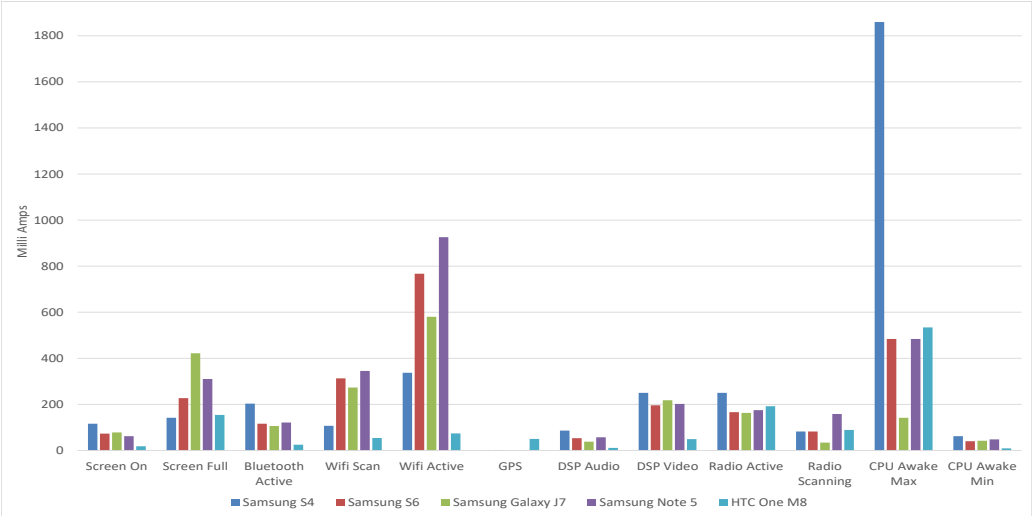


Figure 5: Power Profile for devices listed in Table 1

tions 4.1 to 4.1) we used a Samsung S4 that has a different, possibly more power hungry, hardware composition. As a result, power estimates obtained using PowerTutor were under-estimated for some of the micro-benchmarks.

The Dumpsys utility on the other hand does not use a hand-crafted power model like PowerTutor. Instead it relies on the (OEM provided) power model, which usually ships as standard with Android devices. This information can usually be found in a XML file ($power\_profile.xml$) within the *framework-res.apk* of the device. For instance, we could easily obtain power model information for the devices listed in Table 1. Figure 5 shows a pictorial representation of some of the information contained in these files. According to these XML files, the Samsung S4 display has a significant power consumption. However, the Dumpsys utility reported a near-zero power consumption for the display micro-benchmark (see Section 4.1). On a closer look we observed that the Dumpsys utility did indeed note that the display consumed about 11.3 mAh (power meter reported 13.1 mAh), however, it failed to attribute this power-consumption to the app which was responsible for keeping the display in a high-power state (in this case the display micro-benchmark). In short, PowerTutor generated incorrect estimates because of an outdated power-model, whereas, dumpsys reported incorrect estimates because of incorrect power-accounting.

The results for the power-measurements for the CPU/Memory micro-benchmark for the devices in Table 1 are shown in Table 2. We used PowerTutor in these measurements. Note that PowerTutor reports energy-consumption in Joules, however other measurements (from power meter, dumpsys) were reported in milli-ampere-hour (mAh). Therefore, to maintain consistency across all the presented results we converted the PowerTutor generated estimates to mAh. An interesting observation from Table 2 and Figure 5 is that neither the most power-hungry CPU (Samsung S4), nor the least power-hungry CPU (Samsung Galaxy J7), were most efficient in executing the CPU/memory micro-benchmark. Plausible explanation for such behavior can found in the discussion of Section 2.

Table 2: Energy-consumption of the devices in Table 1 for the CPU/Memory micro-benchmark

Device	Battery Volts	Energy			
		Joules	Wh	Ah	mAh
S4	4.2	114	0.03166	0.00753	7.53
S6	4.197	16.4	0.00455	0.00108	1.08
J7	3.765	615	0.17083	0.04537	45.37
Note 5	4.192	15.2	0.00422	0.00100	1.00
M8	3.897	230	0.06388	0.01639	16.39

5. ENERGY-AWARE PROGRAMMING

The development life-cycle of an application has several stages. Some of the well-known stages of development life-cycle are planning, design, implementation, testing and maintenance. Until recently, the key focus of these life-cycle stages was on or around the functional aspects of the application. However, as mobile apps gain prominence, non-functional properties such as energy-efficiency, are increasingly becoming more important during software development life-cycle. In the following sections, we shall discuss the emergence of an energy-aware programming ecosystem that can address this need. We shall then discuss some parts of this ecosystem that have not received due attention hitherto, why it needs attention from the research community and how it can be provided.

5.1 Energy-aware Ecosystem

Designing for energy-efficiency is not only crucial but also relatively straightforward, in context of mobile apps. This is because in mobile apps a given functionality can be achieved through a number of means, some of which are more energy-efficient than others. For instance, a computationally-intensive task, can performed locally or offloaded to a remote server, based on acceptable performance energy trade-offs and service-level agreements [37]. A data packet that needs to be sent/received from a mobile app can be transmitted/received over either WiFi or cellular radio, both of which have different performance and energy-consumption characteristics. Location updates, in mobile apps, can be gathered using several different strategies, where some of the strategies are less precise but more efficient than others [36, 38].

Even the choice of programming languages can influence the energy-consumption behavior of an app. Consider the work of [39], which proposes a new programming languages ET. The key novelty of ET lies within its new energy-aware, type system. Even

when developing applications using conventional languages such as Java, it is possible to construct a more energy-conscious application. For instance, the work of [40] presents an idea related to approximate-computing that can be useful in context of developing energy-efficient applications. It is possible to improve the energy-efficiency of an app, even after development. For instance, the work of [25,41,42] propose approaches for energy-aware test-generation, re-factoring and optimization, respectively.

One life-cycle stage of energy-aware programming ecosystem that has not received as much attentions as others is the software maintenance stage. It is worthwhile to know that for mobile-apps it is during this stage that apps are used by their intended user and possibly provide financial benefits for its creators. So it is important to provide more attention to this stage of app development life-cycle. Following section discus some of the issues that users complain about while using mobile apps.

5.2 Why do Users Downvote Apps?

Unlike the days of past, when users and app-developers often did not have a direct line of communication, nowadays users can provide immediate feedback to the app-developers, by means of social-media and on app-hosting platforms. Particularly, on app-hosting platforms such as Google Play, app-users can provide feedback of their experience with an app through a star-based rating system as well as their reviews. This feedback provides a mechanism by which a user can *upvote* or *downvote* apps and therefore it may be a key indicator of success or failure of an app in the app store. An user may give a sub-par rating (*i.e. downvote* an app) to an app, if they are discontented with either functional or non-functional aspects of the app. The reason for dis-contention often find its way into the user's review for the app. Therefore, in order to gain insights into the problems that app-users were facing, we created a dataset by looking at approximately $170,000$ Android apps in Google Play store (this is equivalent to about 10% of the apps hosted in Play Store [43]). We also gathered the reviews and stars awarded for each review, for all apps in our dataset. We observed that the user reviews were, in general, short and averaged to about 18 words per comment. We observed that users who were contented with an app provided five (out of five) stars, while less-contented users provide four or less stars to the app. Users also provide a short description of the issues with the app. By manual investigation we were able to categorize these issues into the following five categories:

- *Crash Related*: One of the major issues users report is that the app crashes while in use. Often app crashes are observed (and reported) after an update. Following user review provides one such example:

  AirAsia Mobile, 22 March 2016: *Used to be 5stars but after the last update, app just crashed all the time....*

- *Advertisement Related*: In-app advertisements (Ads) can be a source of revenue for the app-developer. However, excessive or objectionable advertisements can cause a user to *downvote* an app. Following user review provides one such example:

  Utorrent Client, 27 February 2016: *Full screen, full volume ads? I'd give this zero if I could . . .*

- *Performance Related*: Sluggish or unresponsive apps can also be a source of annoyance to app-users. Following user review provides one such example:

  NASA App, 22 Sept 2015: *Nothing loads :(Not even menus load in under minute, . . . It's slow and clumsy*

- *Energy-consumption Related*: Mobile devices such as Smartphones are energy-constrained and therefore must be optimized for energy-efficiency. Especially, energy-efficiency is important for those apps which are designed to run for long durations of time. Following user review provides one such example:

  Automatic Call Recorder, 28 March 2016: *Going bad now This app drained my battery even on idling it only take 20 minutes to drain 10 % of my battery*

- *Other (Functionality Related)*: All other review which do not have five stars and cannot be classified as crash, ads, performance or energy related, fall in this category. Often such issues are due to the functionality of the app. Following user review provides one such example:

  Hulu, 13 April 2016: *Can't watch Won't play over WiFi, saying time and date are wrong*

We used user-ratings as well as keywords to classify the user reviews into the aforementioned categories. Figure 7 (primary vertical axis) shows a logarithmic plot of the issues observed in the user reviews in our dataset. We observed that a substantial number of less-than-5 user-ratings where due to unsatisfactory functional behavior of the app. However, quite interestingly, we observed that a substantial number of user-reviews which were categorized as energy-efficiency related suggested that either the user had or wanted to uninstall the app. User review box 1 provides such an example. To see if such a trend was widespread throughout our dataset we conducted an experiment to check the correctness of Hypothesis 1.

User Review Box 1 : AirWatch Agent

22 March 2016: *This app is the biggest battery hog out of all my apps on my LG G2 If it wasn't mandatory for me to have it installed in order to access my work emails/calendar I would uninstall it immediately. Surprised to see such a piece of junk coming out of VMware*

Hypothesis 1: *A user is more likely to uninstall an app, if it has energy-efficiency related issues as compared to crash, advertisement, performance or other issues*

For this experiment we measured the *Uninstall Ratio* for each type of issue recorded in our dataset, as described in Equation 1.

$$\textit{Uninstall Ratio} = \frac{|Uninstall_{category}|}{|Reviews_{category}|} * 1000 \qquad (1)$$

where $category$ is an element of the set {Good, Crash, Ads, Performance, Energy, Other}. $Reviews_{category}$ is the set of user reviews in our dataset that have keywords for the respective category whereas $Uninstall_{category}$ is the set of user reviews in the set $Reviews_{category}$ that have the keyword 'uninstall' in them. Essentially, *Uninstall Ratio* measures the likelihood that a user will uninstall an app if she/he provides a review that is classified into a given category.

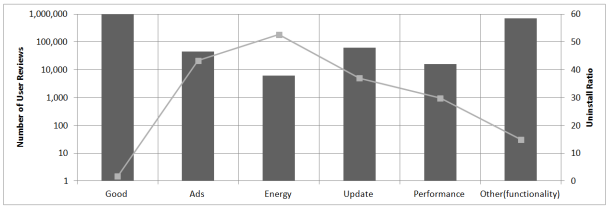


Figure 7: Plots showing the number of user reported comments observed in each category

Figure 7 (secondary vertical axis) shows the *Uninstall Ratio* for each category, using all the comments in our dataset. We observed that indeed energy-efficiency related issues have the highest *Uninstall Ratio* amongst all category of issues. Therefore, Hypothesis 1 one is correct. One plausible reason for such behavior might be that having issues that are not energy related (*i.e.* crash, ads, performance or functionality) only influences the utility of the affected app, whereas, an energy-efficiency related issue can influence the utility of the entire smartphone by drastically reducing its battery life. As a result, app-users may be compelled to uninstall an energy-inefficient app with haste.

5.3 Automated Energy-aware Maintenance

The best-case scenario would be where bug-indicating user reviews can automatically be converted into patches. However, user reviews are usually provided in natural language. Often user review contain clues that may be helpful in identifying suboptimal behaviour in programs. However, using the user reviews verbatim may not be suitable for test generation and debugging. This is because often the user reviews contain a partial description of the temporal ordering between the states (or state transition) within the app. Consider the scenario as shown in Figure 8. The user may observe (and report) that suboptimal behaviour is noticeable at state Z when the execution passes through state X after App Start. However, the user review does not provide information about the intermediate states. For instance, in the example of Figure 8 intermediate states would be states A and B.

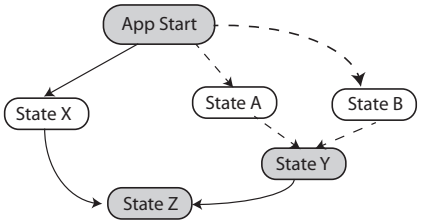


Figure 8: Example state diagram for reference in user review

The challenge here is to device a technique by which user reviews, that are usually provided in natural language, can be represented more concretely, while being flexible enough to accommodate the app-behaviour that has not be mentioned in the review. To address this challenge, we studied user reviews obtained from Github and observed that user reviews that describe the faulty behaviour are generally composed of three types of keywords:

1. *Event Keywords*: This includes keywords that represent physical interactions (by the user) with the UI states of the app or external events (such as change in network strength, acceleration, etc) from the environment. Examples of events are open(app), click, tap, drag, *etc*.
2. *Temporal Keywords*: This includes the keywords that indicate temporal ordering, usually between two or more user events. Examples of temporal keywords are then, until, finally, after, *etc*.
3. *Hardware State Keywords*: This includes words (or phrases) indicating the state of hardware components. Examples of hardware states are GPS is on, Screen is bright, *etc*. It is worthwhile to know that certain hardware states consume more power than others on mobile devices (see Figure 5), therefore keeping track of hardware states may be essential in debugging energy-efficiency related issues.
4. *UI State Keywords*: This includes words (or phrases) that refer to different UI states of the app. Often transition between two UI states occurs as a result of arrival of an event.

Based on these observations, we concluded that if one could design a mechanism which could shortlist useful user reviews based on whether a review describes (*i*) *what* was the faulty behaviour? (*ii*) *how* the faulty behaviour can be re-created? and (*iii*) *which* components (on the mobile device) were affected by the faulty behaviour? it might be possible to develop an energy-aware, automated maintenance framework. Such a framework could use these shortlisted reviews to generate bug-revealing test-cases and subsequently, prepare a patch for the app while alerting the developers. User review box 2 shows an of example of one such user reviews from our dataset.

User Review Box 2: Owncloud, Issue 121

5 December, 2013: *Confirmed Still takes forever to refresh a large amount of files Appears to be pounding the CPU nonstop resulting in rapid battery drain Only way to stop it is to force close the app*

Figure 6 shows an outline for such an automated energy-aware maintenance framework. The framework takes in two inputs, a

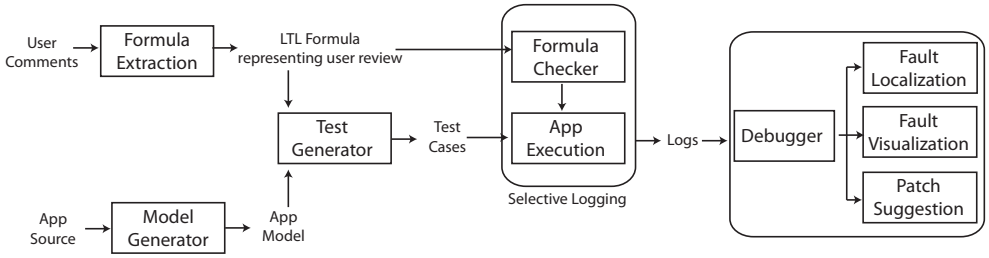


Figure 6: Overview of proposed energy-aware, automated, maintenance framework

Table 3: Tasks, their priorities, task-to-resource mapping and minimum resource costs for the drone

Task Name	Camera (R_c)	Sensors (R_s)	GPS (R_g)	Propulsion (R_l)	Processor(R_p)	Radio(R_r)	Priority
Stabilization, Obstacle Avoidance (T_s)		✓		✓	✓		3
Navigation (T_n)			✓		✓		2
Data Gathering, Processing (T_d)	✓				✓		1
Receive Commands, Send Telemetry (T_c)					✓	✓	2
Recommended Price Per-unit	3	1	2	3	1	2	

list of user reviews for the formula extraction module, and the app source (or *apk*) for the model generation module. The formula extraction module filters and selects the user reviews that have relevant keywords (as described in previous paragraph). The selected user reviews are then converted into a linear temporal logic (LTL) formula ϕ, that has structure as shown in Equation 2. Here symbols X, G, F, U are temporal logic operators representing next, globally, finally and until, respectively. Symbols $\neg, \wedge, \vee$ represent logical connectives compliment, disjunction and conjunction, respectively. Finally, the symbol e represents an element from the set of user event keywords and the symbol h represents an element from the set of hardware state keywords.

$$\phi \rightarrow e \mid h \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \psi \mid X\phi \mid G\phi \mid F\phi \mid \phi U \phi \quad (2)$$

The model generation module automatically explores and generates an UI model of the app and can be implemented as described in one of our previous works [25]. The purpose of the test generator module is to combine the bug-describing, LTL formula with the app model to generate a feasible test-case for the app. Implementing the test generator module might be challenging because the user reviews and therefore the LTL formulae may contain only a partial description of the ordering between the UI states within the app (*cf.* Figure 8). Therefore, the implementation of the test generator module may require the use of suitable search heuristics.

The selective logging module executes the potentially-buggy app on the target device to recreate the buggy-scenarios as described by the user review. However, before executing the app, it is instrumented to log the execution of selected event-handlers and APIs. One of our previous works [33] provides an automated instrumentation tool for Android apps and can be used for this purpose. The logs thus generated can then be scrutinized for the presence of energy-inefficient execution patterns. [33] proposes a set of such energy-inefficient patterns using context free grammar. It also proposes a technique for automated localization, visualization and patch suggestion of energy bugs and hotspots in Android apps. A technique such as this can be used in the last stage of the automated energy-aware maintenance framework to assist with the debugging of reported field-failures.

6. ENERGY MANAGEMENT IN DRONES

The dramatic increase in adoption of mobile phones has been fueled by innovations in both hardware and software technologies. Now, it might be possible that we are witnessing the beginnings of another such growth spike, albeit in a different kind of mobile technology: personalized unmanned aerial vehicles, or *personalized drones* as they are popularly referred to in colloquial conversation. It is worthwhile to know that drones are not a new concept. In-fact, it's origins can be traced back at-least to the early years of the last century (see Hewitt-Sperry Automatic Airplane [44]). However, the concept of personalized drones is relatively new. In the recent years, a number of academic [45], as well as commercial [46, 47] institutions are developing technologies and products related to personal drones. Even though early prototypes of personal drones were merely flying robots and could have been classified as toys, increasingly such devices are being used in more practical applications such as navigation [45], disaster management [48], reconnaissance [49], photography [46], *etc*.

Drones, like smartphones, often face real-time constraints due to the nature of their operations. In addition, these constraints must be met while using the limited on-board battery as the only source of power. As a result, judicious use of on-board battery power is crucial for effective operation of drones. One approach of power-management in such systems would be to create a *centralized* controller which governs the power consumption for all tasks at any given point of time. The key challenge in making such a centralized controller effective would be to encode all possible (power-consumption) scenarios in its design. Failing to do so might lead to unexpected behaviour, including crashes, damage to hardware, personnel or both. However, encoding all possible execution scenarios might be impractical. This is because a drone (or any other cyber-physical system), can go through an exponential number of operational scenarios, on account of it operating in an unpredictable physical environment. The alternative to centralized control would be *distributed* controlled, where power allocation decisions are distributed amongst several tasks that constitute the system. The work of [50] discusses such a distributed power management approach, albeit in context of heterogeneous, multi-core systems. In the following paragraphs, we shall discuss how such a distributed power management approach can be adapted for effective power management on drones.

To keep the subsequent discussion concrete we shall base our discussion on a hypothetical disaster management and monitoring drone which have four basic tasks, that are stabilization and obstacle avoidance, navigation, data gathering and processing and finally receiving commands and uploading telemetry data to the user. These tasks require different resources to operate, as shown in Table 3. In addition, some of the tasks are higher priority than others, as indicated in the last column of Table 3. It is also worthwhile to know that frequency of arrival of these tasks may vary, some tasks such as stabilization and obstacle detection may be more frequent than other tasks, such as receiving commands from the user and sending telemetry data.

The distributed power management approach that we shall discuss in subsequent paragraphs takes inspiration from *Price Theory* in Economics. Very briefly, Price theory [51] provides the basic principles by which the prices of goods and services in the market can be determined by analyzing the supply and demand (for goods and services). Further simplifying the implications of price theory, it can be said that higher demand for goods or services (with supply unchanged), will lead to higher prices. This may lead to lowering in demand or increase in supply. Similarly, lower supply for goods and services (with demand unchanged), will lead to higher prices as well. This again may lead to increase in supply or decrease in demand. The market reaches an equilibrium point when the sup-

ply (of goods and services) matches the demand of agents (who place the demand). The elegance of price theory is that it allows for dynamic price changes based on market conditions. This is the property that we wish to have in our distributed power management approach as well.

Figure 9 shows an overview of the price theory inspired, distributed power management approach. The key idea behind this approach is that there is a virtual *market place* wherein, task agents bid for services provided by resources, using virtual credits, thereby creating *demand*. Resources *supply* the demanded services, in exchange for credits, which they then spend to get raw products, which in this particular case is battery charge. In particular, this approach has three sets of agents.

- *Task Agent*:
  A task agent $t \in \{T_s, T_n, T_d, T_c\}$ (see Table 3) represents and bids for their respective tasks. The bidding is done (every iteration), based on the virtual credits that they have and workload of their respective tasks. Each task has priority (P_t) that is assigned to it by the user.
- *Resource Agent*:
  A resource agent $r \in \{R_c, R_s, R_g, R_l, R_p, R_r\}$ (see Table 3) represents and receives bids for services provided by their respective resources. After a bidding round is complete and credits have been delivered, the resource agent uses the credits to buy battery charge. The battery charge is then used to produce (and supply) the services to the task agents who paid credits for the same.
- *Battery Agent*:
  There is a single battery agent in the system. Its purpose is to maximize the number of completed tasks, while remaining in the permissible battery discharge rate, C_{max}. The battery discharge rate is calculated based on several factors such as, remaining battery charge, expected distance of flight, rate of charge (for drones with solar cells), *etc*. The battery agent also controls the flow of virtual credits in the market.

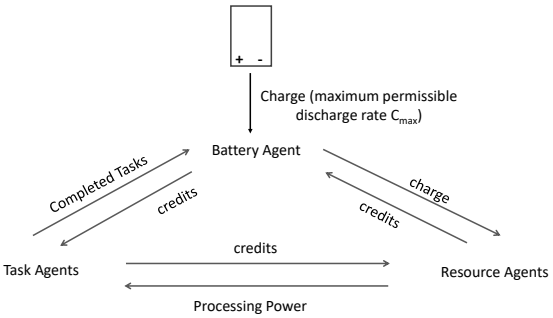


Figure 9: Overview of the distributed power management approach

Bidding. The maximum number of credits that a task agent can bid in round n is restricted to the summation of credits it has been allocated in that round (A^n) and the credit savings it has acquired until that rounds (V_n). A task agent t chooses to bid $B^n_{t,r}$ credits for round n (as shown in Equation 3), by looking at the supply ($S_{t,r}$), demand ($D_{t,r}$) and price (P_r) for the services provided by resource (r). In case the bid in the $(n-1)^{th}$ round was just enough to meet the demand, the task agent keeps the bid for the n^{th} round unchanged.

$$B^n_{t,r} = MIN((A^n + V^n), (B^{n-1}_{t,r} + (D_{t,r} - S_{t,r}) \times P_r)) \quad (3)$$

Price Determination. The price (P_r) for the services provided by a resource r is calculated as shown in Equation 4, assuming that the total supply is S_r. Subsequently, a task agent gets units of services calculated by the formula $\frac{B_{t,r}}{P_r}$.

$$P_r = \frac{\sum_t B_{t,r}}{S_r} \quad (4)$$

Credit Allocation, Savings, Inflation, Deflation. The battery agent distributes credits to each task agent. Unspent credits from a previous can be saved and subsequently used to outbid other agents. However, too much savings may lead to resource-hogging by less power-hungry tasks at the time of emergencies. Therefore, maximum savings per round must be capped (the cap is decided by the user). If during a round any tasks remain uncompleted due to unmet demand, the battery agent may increase the credit allocation in the subsequent round. Allocation of additional credits may cause a task agent to place higher bids, thereby causing price inflation. To combat inflation, a resource agent may increase supply (of services) by switching the resource to a higher power state (say by employing DVFS). In case of deflation, a resource agent may reduce the supply of their respective resources. In the special case where the price for a particular resource falls to zero, the task agent may switch-off the resource, thereby saving power. Finally, in the scenario where battery discharge rate exceeds C_{max}, credit allocation is reduced proportional to the amount by which the discharge rate exceeds C_{max}.

Table 4 shows an example for such a power-management approach in operation. We make the following assumptions in this example.

1. Only task agents T_s and T_d are active.
2. At the start (round 1), per-round credit allocation for T_s, T_d is 12 and 4 credits, respectively. This allocation is proportional to their priorities.
3. Saving are capped at 5 credits per round, per task agent.
4. At the start, all resources are active and are supplying 2 units of services.
5. Resources can only run at discrete power states. Therefore, inflation (deflation) does not cause increase (decrease) in supply until the demand for a resource does not inflates (deflates) to support its next higher (lower) power level. For instance, in this example the processor can only operate at power levels of 1, 2 and 3, producing a processing capacity of 1, 2 and 3, respectively.
6. If a resource is going to increase (decrease) supply in the next round, all bids, as well as, per-task credit allocation are frozen. This is done so that all task agents can see the effects of increased (decreased) supply on the prices.
7. In case tasks have unmet demanded for a given resource r, the battery agents increases the credit allocation by $\frac{D_r - S_r}{D_r} \times B_{t,r}$ for all tasks t.

For the purposes of simplicity we do not show the dormant tasks T_n and T_c and the unused resources R_g and R_r in the example of Table 4. In round 1, task T_s require 1 unit each of sensors, propulsion and processing. Therefore, it places its bids as per the recommended price per unit of these resources (see last row of Table 3). Similarly, task T_d places a bid for 1 unit each of camera

Table 4: An example of distributed power management on a drone

Round	Credits Remaining		Bids					Units Needed, Obtained					Supply, Price-per-Unit			
	T_s	T_d	T_s			T_d		T_s			T_d					
	$A^n + V^n$	$A^n + V^n$	R_s	R_l	R_p	R_c	R_p	R_s	R_l	R_p	R_c	R_p	R_s	R_l	R_p	R_c
1	12 + 0	4 + 0	1	3	1	3	1	1,2	1,2	1,1	1,2	1,1	2 ,0.5	2 ,1.5	2,1	2,1.5
Supply reduced for sensors, propulsion and camera; GPS and radio switched off																
2	12 + 5	4 + 0	1	3	1	3	1	1,1	1,1	1,1	1,1	1,1	1,1	1,3	2,1	1,3
3	12 + 5	4 + 0	1	3	2	3	1	1,1	1,1	2,1.34	1,1	1,0.66	1,1	1,3	2,1.5	1,3
Credit allocation increased																
4	12.7 + 5	4.3 + 0	1	3	3	3	1.3	1,1	1,1	2,1.40	1,1	1,0.60	1,1	1,3	2,2.15	1,3
Processor jumps to next power state, supply increased																
5	12.7 + 5	4.3 + 0	1	3	3	3	1.3	1,1	1,1	2,2.1	1,1	1,0.9	1,1	1,3	3,1.43	1,3
6	12.7 + 5	4.3 + 0	1	3	2.85	3	1.3	1,1	1,1	2,2.06	1,1	1,0.94	1,1	1,3	3,1.38	1,3
7	12.7 + 5	4.3 + 0	1	3	2.7	3	1.3	1,1	1,1	2,2.03	1,1	1,0.97	1,1	1,3	3,1.33	1,3

and processing, as per the recommended price. Observe that due to assumption 4, there is an excess supply of all resources but the processor. Therefore, prices of services provided by R_s, R_l and R_c have deflated below the recommended price in round 1. Although the GPS and radio are not shown in Table 4, the price of the services have also fallen to 0 credits/unit, because no active task had placed a bid for their services. Therefore, in round 2 the supply from R_s, R_l and R_c is reduced to 1 unit. Additionally, GPS and radio are switched off. Also observe that during the supply changes in round 2 all bids and per-task credit allocation are frozen. After round 2 has completed all resources are supplying their services at recommended price, therefore, if no change in supply or demand happens the system will stay in this equilibrium state.

For the sake of this discussion we assume that in round 3 the stabilization tasks needs more processing power, say due to a potential hazard in the flight path. Therefore, it increases its demand for processing services to 2 units. It also increases its bids for processing services to 2 credits, based on the market price of 1 credit/unit from the previous round. However, in round 3, the total supply for processing services is only 2 units, as results a higher bid leads to an inflationary situation, where the price of processing services is ramped up to 1.5 credits/unit, as per Equation 4. Factoring in the new price, T_s gets only 1.34 units of processing services in round 3, while T_d's bid only bought it 0.66 units of processing services. Interestingly, in round 3 since the per-unit price of processing services has not inflated up to the next power level, therefore, the R_p cannot increased the supply in round 4 (as per assumption 5). Therefore, the battery agent steps in, and increases the allocation of T_s and T_d by 0.7 and 0.3 credits, respectively (as per assumption 7). In round 4, T_s bids 3 credits for the 2 units of processing power based on the observed price of 1.5 credits/unit in round 3. T_d which needs 1 unit of processing power can only bid a maximum of 1.3 credits as per the Equation 3. With this new influx of credits, the price of processing services inflates to 2.15 credits/unit in round 4. As a result, the supply of processing services can now be increased to 3 units (see assumption 5). In round 5, the bids and per-task credit allocation are frozen as per assumption 6. An increase in supply of processing service brings down the prices to 1.43 credits/unit at the end of round 5. However, agent T_s is now receiving more processing services than it needs, therefore, it reduces its bids for processing services in subsequent rounds using Equation 3. As a result, after a couple of rounds of bid adjustment, both tasks agents T_s and T_d receive exactly the units of processing services they had demanded for in earlier rounds.

Such a distributed approach of power management is capable of seamlessly integrating several objectives, such as task-priorities, power requirements, fairness, power-saving techniques (*e.g.*, DVFS). In addition, it is also able to adequately react to unpredictable situations that are a common occurrence in real-life drone operations. To highlight these scenarios, in round 2, the unused (less-used) resources were automatically switched-off (scaled back). In rounds $3-4$, a higher priority task was given more resources in times of emergency, even though the supply of resources could not be increased immediately. Also observe that in rounds 5 and later the tasks themselves decided upon the resource allocation as per their respective demands and subsequently came to an equilibrium point where supply equaled the demand for all services.

Concluding remarks. In this article, we studied non-functional properties of mobile software, first distinguishing between performance and energy-efficiency. We discussed some of the challenges in obtaining accurate power measurements, a feature that is crucial for developing an energy-aware programming framework for mobile apps. We presented a proposal for an energy-aware maintenance framework where user reviews on energy issues in an app can be utilized to automatically construct test cases for the purpose of debugging and improving the app. We conclude the article with a forward looking view where we study energy management issues in software for personal drones. Since personal drones will operate in unpredictable environments - energy management requires a flexible on-the-fly approach. We proposed a distributed energy management strategy, using concepts from price theory in economics, for tacking any unexpected scenarios in the operation of personal drones.

Acknowledgements

The work was partially supported by a Singapore Ministry of Education (MoE) Tier 2 grant MOE2013-T2-1-115 “Energy aware programming".

7. REFERENCES

[1] Android version history. https://en.wikipedia.org/wiki/Android_version_history.

[2] Desktop operating system market share. https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0.

[3] List of microsoft windows versions. https://en.wikipedia.org/wiki/List_of_Microsoft_Windows_versions.

[4] Martin Sauter. *Evolution of Mobile Devices and Operating Systems*. 2013.

[5] big.little technology. https://www.arm.com/products/processors/technologies/biglittleprocessing.php.

[6] Yu Lin, Cosmin Radoi, and Danny Dig. Retrofitting concurrency for android applications through refactoring. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 341–352, New York, NY, USA, 2014. ACM.

[7] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 29–42, New York, NY, USA, 2012. ACM.

[8] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM.

[9] Forum-XDA, CPU Governors. `https://forum.xda-developers.com/general/general/ref-to-date-guide-cpu-governors-o-t3048957`.

[10] Shortyz Crossword Game. `https://f-droid.org/wiki/page/com.totsp.crossword.shortyz`.

[11] Paul A. Samuelson William D. Nordhaus. *Microeconomics*. McGraw-Hill, 2001.

[12] What are the actual speeds of GPRS, EDGE, UMTS, HSPA, etc ? `http://www.speedguide.net/faq/what-are-the-actual-speeds-of-gprs-edge-umts-hspa-366`.

[13] What is the actual real-life speed of wireless networks ? `http://www.speedguide.net/faq/what-is-the-actual-real-life-speed-of-wireless-374`.

[14] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Trans. VLSI Syst.*, 2(4):437–445, 1994.

[15] J. T. Russell and M. F. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273)*, pages 328–333, Oct 1998.

[16] Johann Laurent Eric, Eric Senn, Nathalie Julien, and Eric Martin. High-level energy estimation for dsp systems. In *in Proc. Int. Workshop on Power And Timing Modeling, Optimization and Simulation PATMOS*, pages 311–316, 2001.

[17] Johann Laurent, Nathalie Julien, Eric Senn, and Eric Martin. Functional level power analysis: An efficient approach for modeling the power consumption of complex processors. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, DATE '04, pages 10666–, Washington, DC, USA, 2004. IEEE Computer Society.

[18] H. Blume, M. Schneider, and T. G. Noll. Power estimation on functional level for programmable processors. In *Advances in Radio Science*, 2004.

[19] Eric Senn, Johann Laurent, Nathalie Julien, and Eric Martin. Softexplorer: estimating and optimizing the power and energy consumption of a c program for dsp applications. In *EURASIP J. Appl. Signal Process*, 2005.

[20] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: A cycle-accurate energy estimation tool. In *Proceedings of the 37th Annual Design Automation Conference*, DAC '00, pages 340–345, New York, NY, USA, 2000. ACM.

[21] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 28(2):83–94, May 2000.

[22] Doug Burger and Todd M. Austin. The simplescalar tool set, version 2.0. *SIGARCH Comput. Archit. News*, 25(3):13–25, June 1997.

[23] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. Profiling resource usage for mobile applications: A cross-layer approach. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 321–334, New York, NY, USA, 2011. ACM.

[24] Ding Li, Shuai Hao, William G. J. Halfond, and Ramesh Govindan. Calculating source line level energy information for android applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ISSTA 2013, pages 78–89, New York, NY, USA, 2013. ACM.

[25] Abhijeet Banerjee, Lee Kee Chong, Sudipta Chattopadhyay, and Abhik Roychoudhury. Detecting energy bugs and hotspots in mobile apps. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 588–598, New York, NY, USA, 2014. ACM.

[26] Ding Li, Yingjun Lyu, Jiaping Gui, and William G. J. Halfond. Automated energy optimization of http requests for mobile applications. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 249–260, New York, NY, USA, 2016. ACM.

[27] Mian Dong and Lin Zhong. Chameleon: A color-adaptive web browser for mobile oled displays. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 85–98, New York, NY, USA, 2011. ACM.

[28] Bobby R. Bruce, Justyna Petke, and Mark Harman. Reducing energy consumption using genetic improvement. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 1327–1334, New York, NY, USA, 2015. ACM.

[29] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[30] About Samsung Knox Counter. `http://androidfact.com/about-samsungs-knox-counter/`.

[31] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei 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.

[32] Viewing Battery Use Data. `https://source.android.com/devices/tech/power/batterystats.html`.

[33] Abhijeet Banerjee, Hai-Feng Guo, and Abhik Roychoudhury.

Debugging energy-efficiency related field failures in mobile apps. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*, MOBILESoft '16, pages 127–138, New York, NY, USA, 2016. ACM.

[34] Keeping the Device Awake. https://developer.android.com/training/scheduling/wakelock.html.

[35] Receiving Location Updates. https://developer.android.com/training/location/receive-location-updates.html.

[36] Location Strategies. https://developer.android.com/guide/topics/location/strategies.html.

[37] H. Qi and A. Gani. Research on mobile cloud computing: Review, trend and perspectives. In *Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on*, pages 195–202, May 2012.

[38] Abhijeet Banerjee and Abhik Roychoudhury. Energy-aware design patterns for mobile application development (invited talk). In *Proceedings of the 2Nd International Workshop on Software Development Lifecycle for Mobile*, DeMobile 2014, pages 15–16, New York, NY, USA, 2014. ACM.

[39] Michael Cohen, Haitao Steve Zhu, Emgin Ezgi Senem, and Yu David Liu. Energy types. *SIGPLAN Not.*, 47(10):831–850, October 2012.

[40] Woongki Baek and Trishul M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. *SIGPLAN Not.*, 45(6):198–209, June 2010.

[41] Abhijeet Banerjee and Abhik Roychoudhury. Automated re-factoring of android apps to enhance energy-efficiency. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*, MOBILESoft '16, pages 139–150, New York, NY, USA, 2016. ACM.

[42] Irene Manotas, Lori Pollock, and James Clause. Seeds: A software engineer's energy-optimization decision support framework. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 503–514, New York, NY, USA, 2014. ACM.

[43] Number of available apps in the google play store. http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

[44] Hewitt-sperry automatic airplane. https://en.wikipedia.org/wiki/Hewitt-SperryAutomaticAirplane.

[45] Mauro Avila, Markus Funk, and Niels Henze. Dronenavigator: Using drones for navigating visually impaired persons. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*, ASSETS '15, pages 327–328, New York, NY, USA, 2015. ACM.

[46] Staaker. https://www.staaker.com/.

[47] Aridog. https://www.airdog.com/.

[48] M. Erdelj and E. Natalizio. Uav-assisted disaster management: Applications and open issues. In *2016 International Conference on Computing, Networking and Communications (ICNC)*, pages 1–5, Feb 2016.

[49] Florian Segor, Axel Bürkle, Matthias Kollmann, and Rainer Schönbein. Instantaneous autonomous aerial reconnaissance for civil applications. In *Proceedings of the Sixth International Conference on Systems*, pages 72–76. Citeseer, 2011.

[50] Thannirmalai Somu Muthukaruppan, Anuj Pathania, and Tulika Mitra. Price theory based power management for heterogeneous multi-cores. *SIGARCH Comput. Archit. News*, 42(1):161–176, February 2014.

[51] Jack Hirshleifer, Amihai Glazer, and David Hirshleifer. *Price Theory and Applications Decisions, Markets, and Information*. Cambridge University Press, 2005.