Mobile Power: CS5204 Fall 2014 Survey Paper

Ethan Holder
Department of Computer Science
Virginia Tech
eholder0@vt.edu

Due Monday, December 8th at 8:00am

***Abstract*—Modern mobile applications (apps) and their developers face a plurality of challenges, not the least of which is energy efficiency. As smartphones become increasingly pervasive and performant, users in turn demand more energy to run all the apps they desire. While previous research work has been able to achieve marked improvements in mobile energy savings, the approaches utilized are not well-understood outside their respective domains. Furthermore, trends in this field of research are not obvious. This paper aims to correct those problems by surveying current mobile energy saving techniques and their viability. In doing so future app developers may benefit from the techniques presented.**

I. Introduction

This section introduces the motivation for this survey paper along with brief background before enumerating the key research questions and goals and laying out the remainder of the paper.

A. Motivation

The key motivation for this works falls in understanding the current state-of-the-art in energy saving on mobile devices. Many separate groups have seen both commercial and academic success in improving energy efficiency. In order to fully utilize these works and contribute to the research community, researchers must understand what approaches were taken and how effective each was at saving energy. Therefore, this work aims to not only present current research in the field, but also to connect patterns in these works and extrapolate towards future trends. This survey paper should enable future researchers to effectively come up to speed with a variety of research without the need for manually scanning all the details of works currently available.

B. Background

In order to effectively evaluate tradeoffs from one energy saving system to another, a detailed understanding of current state-of-the-art energy saving approaches is necessary. While many varied efforts have seen success in this field, the majority of contributions discussed will fall under the categories referred to here as context sensing, cloud offloading, screen rendering, or energy profiling. Context sensing II-A deals primarily with inferring some user context, typically derived from a combination of hardware sensors, by adapting less expensive sensors or higher level logic. Cloud offloading II-B covers distributing execution of some portion of application code onto cloud servers to reduce CPU load (and thus energy consumption) on the mobile device. Screen rendering II-C addresses issues with expensive graphics operations that can vary the aesthetics the material displayed, but not content. Finally, energy profiling II-D attempts to accurately identify causes of energy drain across levels of abstraction to improve on existing energy saving techniques.

A large portion of mobile energy efficiency research stems from both purely software and hardware techniques as well as mixed approaches. Therefore, this survey will cover research presented under both Software Engineering and Computer Systems. Top conferences across fields, such as SPLASH/OOPSLA [11, 13], ICSE [14, 16], EuroSys [6, 17], MobiSys [7, 15, 18], and HotPower [5, 12, 23], among other scholarly resources [4, 8, 9, 21] are all represented here. Additionally, these works span several top universities and research labs as sources. The papers listed for references in this survey comprise a "top level" set of research work intended to compose a broad spectrum of mobile energy saving issues. Further works in each area will be discussed later in the References subsection III-A.

C. Research Question/Goals

To conduct a successful survey paper, some research question or goal must guide the work. This one aims to address a number of questions, all of which are unified by a single overarching goal. The key research questions are listed below:

- **A)** Firstly, what techniques currently exist to increase energy efficiency/savings?
- **B)** Of those techniques, which are available on what platforms?

C) What available techniques are most effective to utilize and seem most promising in the future?

The larger goal is therefore **to discover and predict emerging patterns in current mobile energy efficiency research**. The Literature Review II and Meta Analysis III sections will largely address the first two research questions. Based on those, the Findings IV section will tie them together in answering the final research question, while the goal should be considered throughout.

D. Paper Layout

The remainder of this paper is structured as follows. The Literature Review II will cover high level areas of research within mobile energy and address the ongoing works in each. The Meta Analysis III will flush out common themes among works and show how similar and different each work is from one another, based on metadata, such as year of publication, amount of citations, venue of presentation, authors, research lab of authors, and research advisor (if available) among other details. The Findings IV will build upon the Literature Review and Meta Analysis to reveal what exactly can be taken away from these works and what trends can be gathered for future works. The Future Work section V will lay out what work could be done to expand upon this survey paper, such as adding more papers, incorporating some other element of analysis, or considering another research field tangential to those presented. Finally, the Conclusions section VI will place the findings in perspective and address what research questions and goals (from the introduction I-C) were answered and achieved in this work.

II. Literature Review

The following section discusses the works studied in this survey. Each paper fits roughly into one of the following large categories, each of which is presented separately: Context Sensing; Offloading; Screen Rendering; and Energy Profiling.

A. Context Sensing

The largest area of mobile power research presented in this survey is broadly defined context sensing. As mentioned above, context sensing deals primarily with inferring some key user state, typically derived from a combination of hardware sensors, by adapting less expensive sensors or higher level logic. These hardware sensors often include accelerometers, GPS units, and wireless radios, such as bluetooth, WiFi, and 3G/4G cellular. By abstracting away the raw data being gathered by these sensors and instead presenting some sort of context API to the developers, many different policies can be applied to limit power usage by sensors while retaining user responsiveness. Below are presented some manual II-A1 and automatic II-A2 methods that abstract several sensors at a time as well as works solely concerning the improvement of wireless radio sensors II-A3.

1) Manual: The Nikzad et al. paper [16] presents a manual approach aimed at context sensing. It requires developers to manually annotate their programs at segments which will potentially consume a lot of energy for whatever reason, largely because of sensor use. These segments can be identifying using other approaches or simple trial and error. The authors have developed their APE middleware to make use of these annotations. Using the annotations along with their middleware, the authors are able to impose varying energy management policies to limit energy consumption across sensors. The authors are able to achieve up to 63% energy savings during execution compared to programs without any policy applied. Although this work presents promising results in energy savings, this system requires heavy manual effort on the part of the developer.

2) Automatic: Automatic context sensing methods offer similar, albeit reduced, energy benefits to the manual versions presented above, but significantly reduce the burden on the developer. This advantage only grows as the mobile market becomes increasingly fragmented, requiring several versions of an application in order to reach the maximum amount of users across competing hardware platforms, operating systems, and application revisions. Beyond this advantage, automatic methods can often learn user behaviors and adapt to differing levels of use. As such, automatic methods of context sensing are preferable in spite of reduced effect compared to manual versions.

The Nath paper [15] applies a layer of abstraction and caching to reducing sensor use and improve energy efficiency. The author presents a mapping between different context states that are mostly mutually exclusive (such as, a user cant be in an “at home” context while in a driving context), but can also show a relationship. Thus, knowing a user is in one state precludes him or her from certain other states or conversely, requires he or she be in a larger, overarching state (such as, if a user is in the “meeting” state, that user must therefore be also in the “at work” state). The author’s system, ACE, derives some states from user data over time, applies a mapping between inclusive and exclusive states, and assigns a priority to sensing each state based on its cost. Thus, when needing one state attribute, traversing these mappings towards the least costly state observation leads to an answer that maximizes the energy savings. The authors are able to reduce context sensing costs by about 4.2 times versus without their system.

The Kansal et al. paper [11] talks extensively about their LAB abstraction for certain hardware sensors, namely accelerometer, gps, and cellular/wifi radio. In this system the developer specifies Latency (time between sensor updates), Accuracy (sensor updates validity), and Battery (drain percentage per day due to application) thresholds then their API figures out how to deliver some context efficiently within the parameters specified. The authors present their Senergy API as an implementation of the LAB abstraction. It provides a better means of automatically sensing context while reducing power usage via those pieces of hardware. Their API shows orders of magnitude improvement over past context sensing approaches. While this approach does require some manual developer input, this work lends itself more to automatic methods since the OS could conceivably apply the same LAB thresholds across all applications as a default and achieve some sort of energy savings as a result with no developer input. This approach also does not require as much manual tuning across application versions as the APE system [16].

The Vekris et al. paper [23] falls partially under Context Sensing and partially under Profiling, which will be discussed more below. In this paper the authors target the existing WakeLock API on Android. The WakeLock API is designed to keep the device on and in an awake state in order to complete some task for typically a limited amount of time (such as keeping data on to finish downloading an update or keeping gps sensors on only while navigating). Using a set of policies they define about when errors can occur with the API, their system keeps track of passes and failures against that policy. Using this, the authors are able to automatically recognize sensors that are staying awake too long using the API and thus document failures.

3) Wireless: The automatic and manual approaches presented above leverage a plurality of sensors to determine context. Those presented in this section focus exclusively on wireless data (cellular 3G, cellular 4G, or WiFi) since this area presents more focused research than any other sensor study.

The Chen et al. paper [5] seeks to reduce energy consumption by eliminating the use of cellular sensors for downloading advertisements. The authors initially measured the energy consumed by downloading ads over cellular network (3G/4G). They selected the top 100 free apps on the Google Play store as a baseline to measure. The authors found that only 3% of energy consumed via the app is spent in obtaining an ad on average (only 57 of the apps tested had any ads). Beyond their limited possibility of savings, reducing that 3% poses significant challenges due to the nature of ad prefetching schemes:

A) They can violate software license agreements.
B) They are not consistent between apps, even on the same platform (platform here mean an Android version).
C) They have limits in how much data (thus how many ads) they can cache.

Their results show that trying to reduce energy consumption via prefetching is not currently viable so long as the behavior of ads across apps remains so diverse, both within a given platform and across a diverse set, and the energy gains remain so minimal.

The Ding et al. paper [8] focuses on reducing energy consumption by only using wireless data in locations with quality signal. First, the authors performed a study to determine how often people go through regions with poor signal. Their large study examined both Wifi and 3G signal from 3785 different smartphones across 145 countries. Using their signal data, they determined that the amount of times people entered regions of poor signal was still significant enough to warrant waiting until reaching a better signal region (that is, poor signal was still encountered enough that is caused a problem, but better signal was still prevalent enough that a user was very likely to reach that region eventually). The authors then determined how much extra cost is incurred on average when in a poor signal region versus better signal. Based on this, the authors formulated an approach for when data transfers can be paused in poor signal in order to reduce the overall cost of all signal transfers. Their new model is system-call-driven and is shown to be able to reduce signal energy consumption by 23.7% on WiFi and 21.5% on 3G.

The Kim et al. paper [12] attacks the problems of tail time in network communications. Tail time is the extra time when wireless sensors are kept on after completing a transfer of some kind. Tail time is necessary to reduce the energy consumed from constantly turning the sensor on and off for every transfer when someone is actively using the device. However, getting the tail time right, such that it captures most cases of active use, but is not so long as to be on unnecessarily (and wasting energy there), is a difficult task. The authors of this work have a system, Personalized Diapause (PD), to automatically extract and profile data about when users are not going to pursue further network communication. They use this data to form an educated, per-user guess as to when the user will no longer be communicating in the next tail time. PD improves over past manual efforts, such as utilizing developer annotations or hints to provide similar functionality, by forming its per-user guess automatically over time. Based on these guesses, it greedily utilizes a fast dormancy feature to reduce the tail time when possible. Reducing the tail time accurately lead to 23% on average (or 36% at max) decrease in overall wireless energy consumption, but also a 10% increase in reconnection overhead (having to reconnect on missed guesses). The authors implemented PD on top of Android 2.3 Gingerbread.

B. *Offloading*

The short Tilevich and Kwon paper [21] is about offloading some "hot" part of execution dynamically at runtime in order to improve energy efficiency. However, network problems get in the way via offloading time and disconnects which impact performance. Therefore, they keep the mobile version running as a slower backup if connection fails. To minimize offloading burden, static analysis with manual program annotations are used, keeping the amount of bytecode offloaded down. They also have some middleware to vary the amount offloaded at runtime based on hardware and network connections. This maximizes energy savings while minimizing the chances of lost connection (useless offloading).

The short Kwon paper [13] deals with mobile energy efficiency in terms of offloading some portion of computation to the cloud. The author shows how we can utilize middleware and dynamic techniques to offload the energy intensive portions of an application to another compute device. This allows us to gain energy savings in terms of computation, but increases network communication. By tailoring the network communication to a given context, the author shows how to further reduce network communication by selecting an appropriate compute device. This combination yields between 25% and 50% more energy savings than state-of-the-art cloud offloading (largely static) techniques. This paper falls more on the systems end of the mobile power spectrum based on its use of program annotations, but it does leverage some unique software methodologies.

The Chun et al. paper [6] is about statically and automatically deciding what to offload from an app and pushing that code to a more powerful server. The authors' system, CloneCloud, is an improvement over past manual annotation approaches (such as [21] above and the Maui system [7]) in that CloneCloud is able to determine upfront what code sections it can offload, and then apply their technique on the fly. The authors built CloneCloud on top of the DalvikVM for use in Android, version cupcake. At the time CloneCloud could migrate unmodified code using their modified DalvikVM to execute on a remote server. By statically determining what to offload and dynamically deciding when, CloneCloud achieves up to 20x improvement in both performance and energy consumption. Although these results are somewhat out of date, the work of CloneCloud still proves the feasibility of a fully automated offloading approach.

C. *Screen Rendering*

The Li et al. paper [14] considers the energy efficiency of web applications relative to how much energy a mobile device's screen consumes. The authors thus propose a means of reducing the energy consumption of html applications to reduce overall screen energy consumption. The authors show how varying the coloring of web pages leads to a decrease in energy consumption of 25% during loading and rendering phases as well as 40% during the display phase. This paper falls on the software end of the mobile power spectrum based on translating an application's aesthetics to a more energy efficient version that is still acceptable to a majority of users.

D. *Energy Profiling*

The Chang et al. paper [4] approaches energy savings by highlighting how existing apps as well as the overall system consumes power. The authors target their analysis on examining the power consumed by the CPU during several distinct cases. Based on their case studies, the authors show how overall performance and energy consumption is impacted when the CPU is performing various tasks, such as video playback, photo editing, streaming video, streaming audio, and social networking. Overall the authors conclude that in their test device, the best performance and energy consumption is achieved when the CPU workload is offloaded to the GPU (in order to maximize idle time and decrease power consumption by the CPU since the GPU is already presumably running). The next best step is to minimize CPU workloads to one core (such that the other can be shut down during execution). The last option to aggressively schedule the workload across all cores in order to complete it faster (again, increasing idle time), but this has a negative impact on battery life overall. Based on their information, further software tuning can be performed at the application or operating system level to improve scheduling and thus increase idle time.

The Ge et al. paper [9] does not focus on mobile device profiling specifically, but this work is currently being extended, so it is included as a motivational work in energy profiling. The authors have developed the

PowerPack system in order to profile energy consumption based on software calls. PowerPack examines the amount of energy flowing through the power lines of the system (cables from the power supply on desktop and battery rails on mobile devices) during interaction by externally tracking software calls throughout. By associating the difference in energy consumption from one unit of time to another with the corresponding software calls that executed in that span, the authors are able to show exactly which calls trigger increases or decreases in power consumption and which components are consuming it. This basic approach is currently being applied to Android tablet devices in order to achieve the same level of granularity from software calls down to the components they affect.

As mentioned above, the Vekris et al. paper [23] falls partially under Profiling, so it bears mentioning that the system utilized there is able to profile the execution of Android apps using the WakeLock API. With their system and warnings 145 apps out of 328 tested were verified to not present a “no sleep” bug. The remaining 183 tested were found to either have the bug or present a warning based on their profiling. This work shows that the results of profiling can have more direct benefits, such as uncovering bugs (especially energy draining ones), rather than just providing a deeper understanding of the system.

The Pathak et al. paper [17] presents a means of fine grained analysis of energy consumption on mobile devices. The authors have instrumented a means of tracking what software calls directly result in draws of power in various pieces of hardware. By utilizing their eprof approach, they have been able to analyze the source code of several popular applications, namely Facebook, Angry Birds, the Google Android Browser, Twitter, and New York Times among others. Further utilizing their energy bundles mechanism, the authors are able to refactor target applications to save up to 65% in energy consumption. This paper falls on the hardware end of mobile power based on their profiling of hardware components and analysis.

The Qian et al. paper[18] demonstrates another means of profiling energy consumption, this time based on layers of interaction rather than calls down to hardware layers. The authors consider hardware components, low-level systems calls, and high-level source code calls in profiling where energy is consumed in a given app. The authors profile several well known applications, such as Pandora, Fox News, and BBC news among others. Their approach is able to identify areas of high energy usage and infer a reasoning for what causes those areas to exhibit such high energy consumption. This paper falls on the hardware end of mobile power based on their profiling of hardware components and analysis.

III. Meta Analysis

The following meta analysis will present common themes between the works surveyed. First, a discussion of the references demonstrates how the works relate and build on one another III-A. Then some commonalities (and potential biases) between works are unveiled III-B. Finally, a comparison of the underlying systems used to implement each technique is presented III-C. This analysis will uncover several avenues for future work.

A. Reference Discussion

First, several papers surveyed here reference others within the survey, building credibility upon each other. For example, Chen et al. [5], Ding et al. [8], Kwon [13], and Vekris et al. [23] all cite the Pathak et al. [17] work in their own. This comes as no surprise since Pathak et al. has been cited 69 times according to the ACM Digital Library [1] and 225 times according to Google Scholar [2]. Additionally, the Pathak et al. [17] and the Ding et al. [8] works both cite Qian et al. [18], which is also well cited itself with 55 and 141 citations, according to the ACM Digital Library [1] and Google Scholar [2] respectively. These works show iterative research progress and their citations demonstrate how well accepted they are within the community, despite being published within only the last 3 years. Finally, Kansal et al. [11] and Nikzad et al. [16] show a similar nature in citing the ACE paper by Nath [15]. Nath only has 18 and 63 citations currently (according to the ACM DL [1] and Google Scholar [2] respectively), but it also demonstrates significant progress nonetheless.

Beyond just these internal citations, the surveyed works also cite numerous other works by Qian, Pathak, Tilevich, Nath, and Kansal as well as those by each’s associated research group or advisor. This only further demonstrates how important these authors are within this field. Several authors not surveyed here also received numerous citations however. Aruna Balasubramanian (and largely the MAUI work [7]), Junehwa Song, Mahadev Satyanarayanan, Andrew Campbell, and Tanzeem Choudhury are all cited multiple times between the works surveyed here. They have each been cited 653, 322, 3641, 2156, and 1116 times respectively, according to the ACM Digital Library [1], for an average between them over 1000. Their research presents a chance for future expansion on this survey based on how prolific each has been so far in their respective careers.

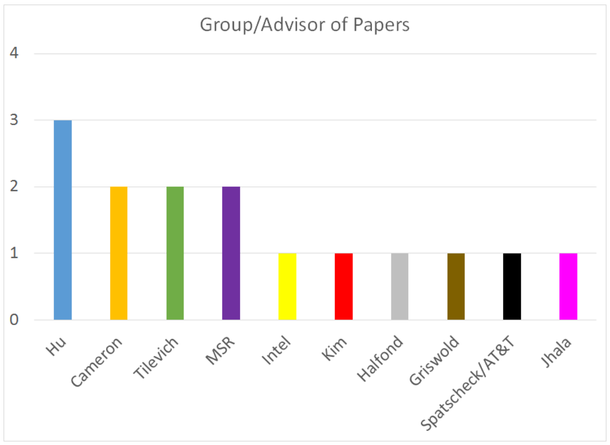

Fig. 1. This shows the amount of surveyed papers by research advisor or company.

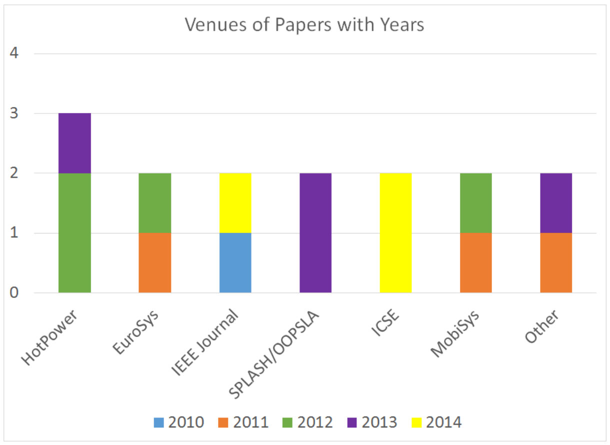

Fig. 2. This shows the amount of surveyed papers at each venue by year.

B. *Commonalities*

Although the papers presented in this survey were intended to cover a wide range of work surrounding energy saving on mobiles, these works do have several notable commonalities and thus potential for bias. First I will talk briefly about the authors, research group, and advisor submitting the papers to show where these have originated. Next I will talk about the venues publishing these works as well as the years of publishing before attempting to identify what the “best” conference for this type of research is.

Every paper presented here has a different first author or primary investigator. As shown in figure 1 however, some of these works originated in the same overall research group. Each bar shown represents one advisor or company (in the case that the work did not come from a university) and shows how many of their works are surveyed here. The largest of them is Dr. Y. Charlie Hu from Purdue. His research group has published numerous other well-cited works surrounding energy profiling and debugging [2] [10]. Dr. Kirk Cameron’s group from Virginia Tech also has several energy-based works in the related field of high performance computing, but has only just begun expanding into mobile devices [3]. Dr. Eli Tilevich’s group (also at Virginia Tech) has worked extensively in automated program partitioning and offloading as well as distributed systems, each with large overlaps on mobile technologies [20].

As mentioned at the beginning of this survey, the works presented herein represent many top conferences across computer science. Figure 2 shows how many of the surveyed works come from each venue (including IEEE journals which are not necessarily related to a conference). The colors of the bars highlight what year each paper was published. As shown all of these works were published in the last 5 years with no single venue producing excessively more than another. This demonstrates that interesting research in this field can in fact end up published in several different venues, yet the data does not definitively suggest one conference over another (since HotPower only has one more paper surveyed than the others, and these papers were chosen without any sort of scientific analysis beforehand).

Upon inspection of the latest proceedings for HotPower (2014) however, it is shown that 5 of the 9 accepted papers as well as both keynote addresses concern mobile energy efficiency in some way. In fact one such keynote address was given by Dr. Hu [22]. Based on this information it would seem that HotPower is clearly focused on this type of work. A more thorough analysis of past years’ publications and their citations in the field since then would likely yield more reliable results, but even this small sample shows promise for this conference. Similar inspections of EuroSys 2014 and MobiSys 2014 did not appear as promising, while IEEE Journals, SPLASH/OOPSLA, and ICSE were not inspected because of the variety and volume of research each publishes annually.

C. *Platform*

One of the key goals of this paper is to determine what energy saving techniques are available for which platform and from that, what platform has the most active improvement. As discussed in the each individual paper, the techniques leveraged to save energy

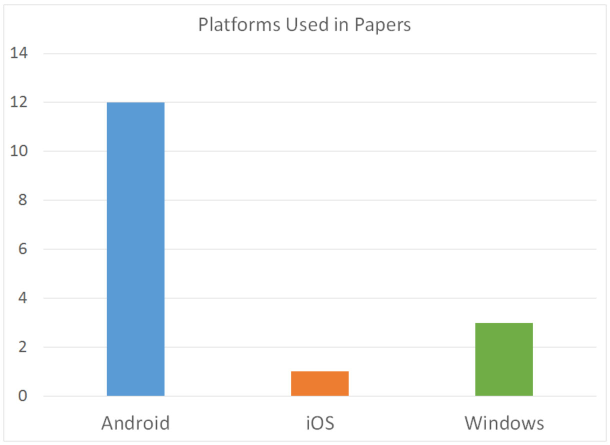

Fig. 3. This shows how many research works surveyed are implemented on each platform.

are largely extensible to any mobile platform. That is, context sensing, offloading, screen rendering, and energy profiling can feasibly be done on any of the major mobile platforms currently available. However, if this survey is representative of the overall research community, then it would seem that the majority of research is carried out on Android devices. As shown in figure 3, the vast majority of research surveyed here is implemented on Android (12 out of 15 papers are implemented on it, compared to 1 and 3 for iPhone and Windows Phone, respectively) [1]. Likely causes of this are both the programmability of Android's underlying kernel based on Linux and the dominance of Android devices in the current marketplace, as shown by a recent Gartner report [19] where Android had 78.4% of the smartphone market in 2013.

IV. Findings

The above literature review and meta analysis sought to answer the first two research goals of this paper, uncovering energy efficiency techniques and their availability. From the literature review, the key techniques in this field seem to be **Context Sensing, Cloud Offloading, Screen Rendering, and Energy Profiling**. Each technique takes a different approach to reducing overall power consumption, but some are more effective than others. The meta analysis showed that although each of these techniques is generalizable to other platforms (and indeed some have already been implemented across a plurality of platforms), the vast majority of surveyed research seems to occur on Android devices 3. It is entirely possible that the results presented herein are an anomaly due to the randomness of the papers selected to survey. However, the current state of the smartphone market [19] suggests that Android devices are indeed the majority and that this trend will generalize to further research not surveyed here.

Based on the results of sections, I now aim to answer the final research question: what techniques are most effective now and most promising in the future. The most obvious answer is to implement all of these techniques in some fashion. However, for developers limited to one approach based on time, cloud offloading seems to be the best technique. There are many approaches available to offloading (even though only a handful is presented here), and that technique has plenty of cross cutting history with distributed systems problems, so the area is well defined. From the results presented here, CloneCloud [6] alone showed up to 20x improvement in both performance and energy when implemented on Android. This technique in particular is automated so the developer burden is further reduced. A close second to cloud offloading would be automatic context sensing, since the results there are also automated and somewhat promising. However, cloud offloading seems to be the more automated of the two techniques at the moment and thus easier to implement. For this reason, automatic context sensing looks to be more promising in the future as more research flushes out best techniques and practices. As discussed with the LAB abstraction paper [11], future work could predefine parameters for how responsive and accurate an application must be, either from the OS or a power management perspective, and then iterate over the cost and benefits to derive more savings from the Senergy system.

V. Future Work

Based on the results given in the Meta Analysis section III, there are numerous ways that this work can be expanded. Due to time and paper limitations however, those expansions are left to future work. The most obvious direction to go is the addition of more research works. The References subsection III-A showed that outside works by the authors already surveyed could present an interesting avenue to expand upon. I hypothesize that those outside works would not add much new information about each technique presented, but they would instead deepen understanding of the foundations for each technique and additional implementations of them.

Another potential expansion of this work lies in exploring more authors. As previously discussed, Aruna

[1]Some of the research works surveyed are implemented on multiple platforms and so are included with each, while there are also some that are not implemented on any platform specifically.

Balasubramanian, Junehwa Song, Mahadev Satyanarayanan, Andrew Campbell, and Tanzeem Choudhury all seem to be well cited and have several works referenced in the papers surveyed. Some of these works, such as [7], present large amounts of citations on their own, further indicating their worthiness of further investigation.

Further analysis of the web of references from surveyed papers and beyond could present more candidates to add. Formalizing this analysis by inputting each survey paper as a leaf note into a graph and tracing backwards through the works each references could uncover the core works within this area. Although every paper will likely not converge to any one source, the papers representing each technique discussed here will likely converge at a few root papers that began this research on mobile devices. This root research may not present much novel information, but it is still important to reference the history that led to each of these works so as to understand what may come in the future.

Yet another avenue for future work lies in specifically targeting platforms besides Android. As discussed previously, the vast majority of work surveyed here is implemented on Android. Although the works surveyed were selected without intent to cover one platform more than another, this could potentially lead to some bias. By specifically seeking out further works solely targeting iOS, Windows Phone, or other platforms, there is a chance to uncover new techniques that may not be applicable to Android.

VI. Conclusions

Many research projects have been able to reduce energy consumption on mobile devices. By reducing the energy drain of context sensors, offloading intense computation to the cloud, adjusting the aesthetic rendering of an app, or profiling the overall consumption of energy each work surveyed here achieved marked improvement either in energy savings directly or in finding energy problems which could lead to future energy saving opportunities. These techniques are extensible to most major platforms in principle, but the works surveyed here were largely implemented on Android. The most promising work in the future could likely be from Dr. Y. Charlie Hu's group or from the MobiSys conference based on the data obtained here. Future additions could explore potential sources of bias by broadening the amount of surveyed work.

Acknowledgments

I would like to thank my CS 5204 Operating Systems instructor, Dr. Ali Butt, and fellow classmates for all of their support to complete this work and theirs evaluation at intermediate progress reports and presentations. I must also thank Dr. Kirk Cameron for his inspiration to undertake this work as well as for his recommendation of a few distributed systems papers to get me started. Last but not least, I would like to thank my advisor, Dr. Eli Tilevich, for his continuous guidance and tutelage throughout all of my work. Without all of them this work would not be possible.

References

[1] Acm digital library. http://dl.acm.org/, Dec. 2014.

[2] Google scholar. http://scholar.google.com/, Dec. 2014.

[3] K. Cameron. Kirk cameron. http://scholar.google.com/citations?user=ZH1pa0EAAAAJ&hl=en, Dec. 2014.

[4] H.-C. Chang, A. Agrawal, and K. Cameron. Energy-aware computing for android platforms. In *Energy Aware Computing (ICEAC), 2011 International Conference on*, pages 1–4, Nov 2011.

[5] X. Chen, A. Jindal, and Y. C. Hu. How much energy can we save from prefetching ads?: Energy drain analysis of top 100 apps. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, HotPower '13, pages 3:1–3:5, New York, NY, USA, 2013. ACM.

[6] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[7] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[8] 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.

[9] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658–671, May 2010.

[10] Y. C. Hu. Y. charlie hu. https://engineering.purdue.edu/~ychu/, Dec. 2014.

[11] A. Kansal, S. Saponas, A. B. Brush, K. S. McKinley, T. Mytkowicz, and R. Ziola. The latency, accuracy, and battery (lab) abstraction: Programmer productivity and energy efficiency for continuous mobile context sensing. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages &#38; Applications*, OOPSLA '13, pages 661–676, New York, NY, USA, 2013. ACM.

[12] Y. Kim and J. Kim. Personalized diapause: Reducing radio energy consumption of smartphones by network-context aware dormancy predictions. In *Presented as part of the 2012 Workshop on Power-Aware Computing and Systems*, Berkeley, CA, 2012. USENIX.

[13] Y.-W. Kwon. Orchestrating mobile application execution for performance and energy efficiency. In *Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, &#38; Applications: Software for Humanity*, SPLASH '13, pages 125–126, New York, NY, USA, 2013. ACM.

[14] D. Li, A. H. Tran, and W. G. J. Halfond. Making web applications more energy efficient for oled smartphones. In *Proceedings of the*

36th International Conference on Software Engineering, ICSE 2014, pages 527–538, New York, NY, USA, 2014. ACM.

[15] S. Nath. Ace: Exploiting correlation for energy-efficient and continuous context sensing. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 29–42, New York, NY, USA, 2012. ACM.

[16] N. Nikzad, O. Chipara, and W. G. Griswold. Ape: An annotation language and middleware for energy-efficient mobile application development. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 515–526, New York, NY, USA, 2014. ACM.

[17] A. Pathak, Y. C. Hu, and M. 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.

[18] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. 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.

[19] J. Rivera. Gartner says annual smartphone sales surpassed sales of feature phones for the first time in 2013. http://www.gartner.com/newsroom/id/2665715, 02 2014. Accessed: 2014-05-09.

[20] E. Tilevich. Eli tilevich. http://scholar.google.com/citations?user=t4OSUfgAAAAJ&hl=en, Dec. 2014.

[21] E. Tilevich and Y.-W. Kwon. Cloud-based execution to improve mobile application energy efficiency. *Computer*, 47(1):75–77, Jan. 2014.

[22] USENIX. Workshop program. https://www.usenix.org/conference/hotpower14/workshop-program, Oct. 2014.

[23] P. Vekris, R. Jhala, S. Lerner, and Y. Agarwal. Towards verifying android apps for the absence of no-sleep energy bugs. In *Presented as part of the 2012 Workshop on Power-Aware Computing and Systems*, Berkeley, CA, 2012. USENIX.