Literature Survey to study Energy Optimization Techniques in Mobile Apps

Nishtha Chawla, Ravneet Kaur

1. INTRODUCTION

The smartphones have become an inherent part of our lives and they have completely changed the way people interact. They make the life easier because they assist us in performing the variety of jobs. Apart from using a mobile phone for calling purposes, it is widely used for accessing and processing multimedia data, transmitting data over the internet, using GPS sensors to find locations, playing games, sending emails, editing documents etc. By 2020, the number of mobile device users is predicted to reach 6.1 billion [1]. With the rapid increase of mobile devices, the apps running on them are increasing manifold. As of March 2017, Android's play store had 2.8 million apps and Apple's app store had 2.2 million apps. Mobile apps offer simplicity in performing the daily chores, like shopping, booking tickets, purchasing grocery, paying bills and many more. Despite the faster CPU's, sophisticated networks and larger memory, still, the utility of mobile phones is limited by their battery life [2]. Even when the app and the screen are turned off, the applications may still consume some energy in the background, without the user being aware of it. We have smartphones, but not so smart battery life. The usability of the mobile phone is restricted because they have a limited battery life which confines the services and benefits available to the users. So, optimizing the energy consumption of the mobile apps can allow users to fully utilize their mobile devices and take advantage of the functionalities provided by these apps [3].

The energy is drained unnecessarily in numerous forms such as tail energy, network energy, energy bugs and a connected standby state. Several types of research have been undertaken to optimize mobile app energy by reducing the energy consumed by components such as OLED, Wi-Fi, CPU, GPS etc. A large amount of battery is drained by the ads on the mobile app, even when the user is interacting with the app without the network usage, the ads still consume the network data[13]. The mobile web apps can also be customized to make them energy efficient for OLED Smartphones, by program analysis of the structure of web app implementation[7]. Banerjee et. al. proposes an automated test generation framework that generates test inputs which capture those user interactions that lead to energy hotspot/ bug in an application[15]. Another approach to reducing the energy consumption of an app focuses on saving network communication energy at app level by detecting and bundling HTTP requests and transmitting the bundled request over the network[4]. Another approach to solving energy efficiency problem in Android apps is by detecting resource leaks which lead to huge energy consumption[14].

Energy consumption is an important quality parameter and it impacts the usability of the mobile app. To the best of our knowledge, we could not find any comprehensive study which collectively investigates various energy optimization techniques at the app level. With this motivation, we have performed a

Literature Review to study, review and analyze numerous energy optimization techniques for the mobile apps.

In this paper we describe our work as follows: Section 2 describes the methodology we followed to completer the literature review. Section 3 describes the categories, Section 4 describes the key points identified to discuss, Section 5 gives the relation to our work to the existing one, and Section 6 we have described the contribution of each team member.

2. METHODOLOGY

We have performed a Literature Review on the techniques to optimize energy efficiency in mobile apps. We looked at the main proceedings and workshops of the conferences year by year from DBLP which is a computer science bibliography website that provides information on major journals and proceedings. We surveyed 9 major conferences for years 2013-2017 to screen potential research papers related to the optimization of energy efficiency in mobile applications.

Below is the list of conferences surveyed to extract the research material for the literature review.

S. No.	Conference Name	
1.	ICSE	*International Conference on Software Engineering*
2.	ASE	Automated Software Engineering
3.	ISSTA	International Symposium on Software Testing and Analysis
4.	FSE	Foundations of Software Engineering
5.	NDSS	The Network and Distributed System Security Symposium
6.	MobiSys	The International Conference on Mobile Systems, Applications, and Services
7.	MobiCom	International Conference on Mobile Computing and Networking
8.	OSDI	Operating System Design and Implementation
9.	USENiX	USENIX Annual Technical Conference

Post the conference survey we shortlisted 23 workshop papers and 49 research papers from the above-mentioned conferences as potential studies for our Literature Review. We documented the shortlisted papers into a spreadsheet and listed their respective conference name, year, authors and paper title. We then proceeded to filter between relevant and irrelevant papers based on the scope of our literature review. We excluded those papers which focused on system level optimization or hardware level optimization and included only those papers which performed energy optimization for mobiles at the application level. We dropped the workshop papers from our review due to time constraints and confined the literature review focus only on research papers from main proceedings of the mentioned conferences. We have listed the reason of exclusion for each irrelevant paper removed from the scope of our review in the spreadsheet.

Above process yielded 24 relevant papers and we categorized each paper into a section and assigned a label for the section to every paper (5 sections being: M=Measurement, T=Testing, L=Language, F=Functional, E=Empirical). We limited the scope of the review to functional and empirical energy efficiency research papers for mobile apps from 9 conferences during the years 2013-2017. The final set of study thus comprised of a total of 15 functional and empirical research papers for the literature review. The list of papers used for the purpose of this literature review are included in Appendix A. The workshop papers whether related or unrelated to the subject of our study are not included in our literature review are listed in Appendix B, and the other irrelevant papers are in Appendix C at the end of the report.

As the next step of the approach, we divided the research papers in the team for their detailed and review. We read and summarized the allocated papers. We organized and categorized the papers into different sections based on the identified criteria "What is optimized". The goal is to present an unbiased review of the works done and not advocate any specific technique to achieve the desired target. The list of the categories is mentioned below.

A. Network Communication Energy: This category discusses techniques proposed by researchers which aim to optimize the energy of an app during network communication. The reviewed techniques have a common goal to reduce the energy consumed by the app while interacting with the network. Major examples of such energy consumption include tail energy and energy used by the app while making HTTP requests.

B. Graphical User Interface: This category lists techniques proposed by researchers which aim to reduce the energy consumption of GUI's in Android apps. In this case, GUI energy consumption under different conditions is being optimized through a variety of methods such as implementing new color schemes to reduce the energy cost of GUI, replacing the use of microphone with other sensors (such as an accelerometer which consumes less energy) to recognize voice input.

C. Background Activities: This category mentions techniques that target to solve the problem of battery drain due to app-originated background activities (sync, wakelocks etc.) that wake up the mobile system which leads to more energy usage. During the screen-off interval, the energy is still consumed due to the background activities such as app synchronization or refreshing the app state.

D. Power Hungry Device Components: This category highlights techniques which focus to reduce the power consumption of certain targeted device component or a group of device components. This includes resources such as Camera, GPS, Radio, CPU, Screen etc.

E. Network Communication & Background activities: This category focuses on techniques that address energy issue arising due to background activities which involve significant network interaction. Background activities that proactively connect to the network to communicate with servers consume a substantial amount of power which leads to quick battery drain. This category details solution which is devised to solve the above problem.

3. DESCRIPTION OF CATEGORIES

A. NETWORK COMMUNICATION ENERGY

Network communication is one of the significant energy consuming operation in a mobile app. The techniques discussed in this category optimize network communication energy of an app at application level instead of hardware or system level.

Li et. al. proposes an automated approach to bundle the sequential HTTP requests of an app and thus reducing energy consumption. To detect the number of HTTP requests in a method they use static analysis to identify the sequential HTTP requests. String analysis is then used for identifying components of HTTP request such as values of variables used as arguments to API invocations[4]. The app is modified by replacing original HTTP API with Agent HTTP API which decides the bundling operation. Requests are redirected to the proxy instead of server and Proxy carries out server side of optimization. The proxy has bundlers which handle unpacking and redistribution of bundled requests. This technique achieved energy saving at the whole application level of 21% for Wi-Fi, 15% for LTE and 8% for 3G.

A large amount of network energy is drained in loading and streaming the video. In this case, the network energy is wasted either by prefetching the content and the user might not watch by just abandoning it or the energy is wasted as the tail energy, which is the energy consumed to keep the radio on for the specified time after receiving small content. eSchedule algorithm proposed by Hoque et.al. collects and analyze the crowd-sourced viewing statistics like the audience retention graph for YouTube, to estimate the user's behavior, and determining the optimal download schedule[5]. eSchedule algorithm takes tail energy into account, downloading the large chunk of the video if tail energy is large and keeping the video chunk size small if tail energy is small. eSchedule algorithm reduces the energy network energy waste to approximately half.

Bui et. Al. reduce the energy consumption of web page loading on the mobile browser without increasing page load time[6]. They modify the Chromium version 38 for energy efficient web page loading. The technique performs batch processing of web resources to reduce overhead on small data sizes. A trade-off between energy saving and delay is achieved by increasing batch size on the fast network to save more energy and decreasing it on the slow network to reduce delay. Adaptive content painting is implemented to aggregate multiple content paints to reduce the unnecessary computation of small visible changes paints. This gives a trade-off between user experience(UX) and energy as a low frame-rate saves energy but has worse UX. Application-assisted scheduling (AAS) technique leverages internal knowledge of the browser to make better scheduling decisions. Instead of OS they allow browsing to decide the thread assignment too big or little cores. Total system energy consumption on Galaxy S5-E phone is 24.4% and 11.7% on Galaxy S5-S.

B. GUI

Smartphones provide end users with a range of sensors that can be combined with applications and data via the Internet. This makes the capabilities of smartphones almost boundless and very popular with end users thus enhancing the GUI. OLED screens are used in every mobile phone these days are more energy efficient than previous generation displays, but also have very different energy consumption patterns[19].

Li et. al. advocates a technique of program analysis on the statements of the web app and re-writing the web pages are the server side to generate the more efficient web pages and displaying them on the smartphone via automatic redirection or a user-clickable link[7]. Nyx(the program analysis tool), analyzes the HTML elements of the webpage and identifies relationships among elements in the form of a CFG(Control Flow Graph). The color scheme of the elements is replaced, maintaining the same color distances as the original graph and the color of the text is also replaced. In the final step, the web pages are written according to the new color scheme. Nyx reduces the energy consumption of OLED by an average of 40%.

Vasquez et.al. propose to minimize energy consumption by choosing appropriate colors and color model. They introduce an approach named GEMMA to generate color compositions which reduce energy usage by GUI in Android apps[8]. GEMMA is a combination of power models, color theory, dynamic analysis and pixel-based engineering. The output solution is in form of a Pareto optimal set of GUI color compositions. Optimization is achieved by obtaining power model of the display by relating pixel color to power consumption followed by identification of GUI elements which compose each screen and in the last step a multi-objective genetic algorithm is defined to find optimal solutions. A power model for SUPER AMOLED screen is built using Monsoon Power Monitor and dynamic analysis is used to extract GUI-based information from an app. Energy consumption Fitness (ECF) computes estimated consumption by taking the sum of power consumption of all pixels in screen and weighing them for an estimated fraction of time for the screen in use. 3 GUI per app was considered for the optimization process. For each app, GEMMA was run 30 times. Lowest ECF ensures mean energy saving up 66% i.e. most energy-efficient color schemas recommended by GEMMA consume one-third of the original color design[9].

Another GUI component which significantly increases the energy consumption of the smart-devices is the microphone, used for voice control apps. These apps continuously monitor user's voice input through the microphone continuously listening for hotwords such as "Okay Google" or "Hi Galaxy" thus increasing the battery spent. Accelword, a voice recognition application, proposed by Zhang et. al., is based on accelerometer sensors to detect the hotwords and turns the microphone on only when required[10]. The offline training of Accelword is done to construct the signature dictionary consisting of hotwords and a machine learning classifier performs hotword detection. Using the accelerometer sensors save the phone's battery life as it does not keep the microphone switched on all the time waiting for user's command. The accelerometer's sensors can analyze high-frequency signals and can also detect voice signal during walking, running etc. The results of Accelword app demonstrate that true-positive rates of Accelword are 99.1% and 97% of true-positive rates of Google now and S voice.

C. BACKGROUND ACTIVITIES

The apps may periodically wake up the OS even when the users are not interacting with apps, just to refresh the state or to synchronize with any cloud services or to support non-touch user based interactions. These are the background activities which leads to resource and energy usage even when the phone screen is turned off.

Martins et. al. investigates matters of battery exhaustion due to background operations originating in the app that wake-up mobile system[11]. They implement the solution as an OS mechanism, Tamer, for

Android that interposes on events and signals which cause task wake up such as alarm, wake locks etc. The solution is implemented as a configuration mechanism which declares threshold to the frequency of events that keep the system awake. A policy mechanism allows the user to define the permitted number of occurrences for a background event to be allowed for execution. Event throttling is achieved by canceling or delaying the event. 3 different policies of location update of every 1, 5 and 15 seconds are tested for Galaxy Nexus and it leads to saving 27.7 % energy when location update rate is moved from 1 to 15 second. Tamer intercepts event happening, inspects it, evaluates policy and finally actuates to fulfill policy condition. It also incurs overhead because it diverts normal application flow.

The research by Chen et. al., analyze the behavior of the apps and examine the battery used by these apps in the background[12]. The background activities improve user experience only if they are used by the user in the foreground. So, the background-foreground correlation using BFC metric helps in knowing from past data which apps are not used by the user in the foreground and suppress those apps to reduce their energy drain. “HUSH”, screen off Energy optimizer, monitors the metrics online and automatically identifies and suppresses background activities during screen-off periods that are not useful to the user experience. HUSH saves screen-off energy of smartphones by 15.7% on average while incurring minimal impact on the user experience with the apps.

D. POWER HUNGRY RESOURCES (CPU, Wi-Fi, GPS, SENSORS)

Guo et. al. solves energy efficiency problem in Android apps by detecting resource leak problems i.e. situations where resources should be released manually by the developer but they are not. Resource leaks lead to high energy and memory consumption. The focus is on resources which need explicit releases such as exclusive (camera) resource, energy consuming resource (sensor) and memory consuming resource (media player component). Static Analysis is used to automatically analyze resource operations of an app. Bytecode is traversed to construct a Function Call Graph which has an analysis of resource operations and callbacks[14]. Depth-first search on FCG identifies unreleased resources. For each app, the tool, Relda, outputs resource leak items and logs files details about the request and release of resources as a resource leak summary report. 81 leaks were found in true positives, 11 in false positives and 8 in false negatives.

Banerjee et. al. detects energy bugs and energy hotspots in mobile apps to make mobile app development more energy efficient[15]. They propose an automated test generation framework which generates test inputs that capture user interaction which lead to energy hotspot/bug in an application. The approach is modeled on direct power measurement instead of an energy model. A search strategy is used to generate user interaction scenarios which invoke system calls that access input-output component. A pool of system calls is created which is used to access components like Wi-Fi GPS. Load and energy consumption is done for screen, CPU, Wi-Fi, radio, and GPS. Energy consumption to utilization ratio is measured for a given period to compute the magnitude of energy inefficiency of an application. Energy bugs were detected for 10 out of 30 programs and hot spots were reported for 3 programs. To reduce false positives, the event trace was executed manually which revealed one false positive. The outcome is test report with the set of test cases describing energy issues. Each resulting test case has energy trace pattern, details of energy issues, set of invoked calls and script for execution of events.

TOFU (turning off UI elements), proposed by Huang et. al., advises removing the code of UI elements that are not desired by the user to reduce the battery consumption in smartphones[16]. The proposed technique does not operate on the app’s source code, so this can prove beneficial for the apps where

source code is not available. All the UI elements in the source code are identified and for the API methods calls which display specific UI elements, the data and variables associated are identified. After discovering all the elements, removable ones are removed iteratively. After removal of undesired code, the data, battery, CPU and Wi-Fi usage is found to be reduced by 48.5%, 92%, 90.2% and 42.9% respectively.

E. Network & Background

Xu et. al. analyses power usage and performance of email sync connected standby only on cellular network for windows phone and Android OS[17]. Email sync operation wakes up the smartphone for data reception and processing. To measure the power consumption of smartphone, monsoon power monitor was used and energy cost was measured without any user interactions and with screen off. Energy cost on Android is higher than Windows phone because Windows has 5 second shorter 3G tail time. To reduce energy waste from 3G tail time the authors suggest that OS should provide global service to connect network usage information of applications running in standby to decide a shortest 3 G tail time and put 3G interface to sleep mode as quickly as possible. Energy cost can be reduced by receiving new email over continuous network connection because client maintains a consistent network connection with the server to receive notification of new email. To decouple data processing from data transmission, the approach suggests batching of transmission together by the email client. Major contributing factors in energy saving are reducing the 3G tail time and reusing TCP connection and both these are independent of incoming email size and inbox size.

4. DISCUSSION

The HTTP bundling technique does not need any new protocol or language. It is independent of system infrastructure and saves time in network transmission because there is less number of requests to be processed[4]. The major flaw in this technique is that Proxy would be needed for each server which doesn't make the idea feasible. The idea to modify browser to save energy during mobile web page loading has a high standard deviation in results as the total energy savings for same technique vary significantly for different devices and different websites ranging from 2 % to 66 % and the values vary from handset to handset as well.

GEMMA gives different possible solutions and allows developers to choose from the set of color solutions[8]. Colors are optimized for multiple screens and duration of the display time of each screen is taken into consideration. It proposes solutions with colors that are not too distant from original ones but does not consider GUI color schema as valid in which adjacent elements have the same color or adjacent colors with low contrast to promote readability of GUI. Changing the colors of the mobile web apps from light to dark may not be always pleasant for the users. User acceptability is hampered and the app ratings. Accelword does not have any mechanism to cancel the effect accelerometer caused by the audio noise[10].

The approach to tame background activities just dismisses or cancels the event when there is a need for throttling. They can be synchronized better or processed in batches. However, this would need precise tracking of events and proper handling of old values. Event frequency approach might not be applicable

to Microsoft Windows or Apple devices as they have different policies about background tasks such as windows phone have quotas to resources like CPU, memory, network usage[11]. The approach to suppress unnecessary background activities is a better solution than the android built-in feature “Restrict Background Refresh”, which prevents all the apps from running in the background altogether. HUSH suppresses only those apps which are not required by the user in the foreground. The HUSH app is not evaluated for any real human users as the experiments are conducted in a controlled way and do not consider the technique how it would be used by real users[14].

The approach to detect resource leaks considers only globally declared resources and assumes that for resources declared in a function, the developer will release them when the method finishes[14]. Energy bug detection approach has a major limitation because the event flow graph may not cover entire application so test generation method is incomplete since all sources of energy hotspot or bugs may not be exposed. Also, search focuses on testing input-output operation only and does not consider certain CPU bound events that drain energy. UI driven app customization technique can be potentially used in the last stage of app development to generate many customizations to meet different needs. It does not require source code, and the legacy apps whose source code and original developers are no longer available can benefit from this technique. The paper addresses removal of unwanted UI elements which are provided by the app providers, not the third-party advertisements.

The approach to coalesce events to save energy in the context of email sync seems promising. Rather than the cancellation of events, combining or better scheduling of events is a better approach to save energy of handling events. Also, the technique is applicable to other applications which receive push notifications. The total energy savings range from 41 to 47 % so the evaluation shows less deviation[17].

The solution proposed by the majority of techniques to optimize network communication energy is to bundle the requests or process network requests in batches. The common hurdle they face is the network adaptability and managing server interactions. Energy optimization schemes with regards to GUI propose alternative color schemas but the biggest issue is user acceptability when it comes to changing colors of a mobile app to save energy. The approaches to control energy usage by background activities propose to delay or cancel them but this approach also poses usability issues.

As a part of our survey, we included two papers described below with which we are not convinced that these papers were fully relevant and appropriate in the context of our survey. So, we have not categorized them in subsections above but the approaches and the reasons for their exclusion from the survey have been discussed below.

The research by Gui et. al. conducts the systematic investigation of the mobile ads which are provided to the user at the UI of the app[13]. These unwanted ads consume a lot of energy without the user’s knowledge, so they are termed as hidden costs. The mobile ad may load ads from a remote server using the network resources, for which many users are billed by the number of bytes; loading and rendering ads requires CPU time and memory, which can slow down the performance of an app; and finally, these activities require battery power, which is a limited resource on a mobile device. Monsoon Power Meter is used to measure the energy consumed by both the versions of the app, with and without ads and their statistics are recorded. The statistics of energy consumed by the individual components are not specified

in the paper, nor the network energy which is optimized by removing the ads fetching content over the network is addressed.

Liu et. al. focuses on finding and characterizing the performance bugs which can badly hamper the performance of the smartphone and slow down the applications or consume extra resources. The authors conducted an empirical study to classify the bug types and its impact, how these bugs manifest themselves and studied the common causes patterns of these bugs[18]. PerfChecker, a static code analyzer is used to detect the performance bug patterns. From the experiment set of 29 Android applications, 126 instances of performance bug patterns are identified by PerfChecker. But how serious are these performance bugs, and the battery drainage contributing to the presence of performance bugs is not identified.

5. RELATED WORK

Ahmad et. Al., Application energy profiling schemes exploit either hardware-equipment or software-based solutions to track battery-draining behavior during application execution in mobile devices[20]. This study comprehensively reviews state-of-the-art mobile application energy profiling schemes to investigate the strengths and weaknesses of existing schemes.

Perrucci et. Al., In this survey, the energy consuming entities of a mobile device such as wireless air interfaces, display, mp3 player and others are measured and compared[21]. The presented measurement results allow the reader to understand what the energy-hungry parts of a mobile device are and use those findings for the design of future mobile protocols and applications.

Jiucai et. Al., In this paper, we review the recent advances in power-aware mobile multimedia, especially the adaptation technologies applied in video coding and delivery[22].

Kumar et. Al, Mobile systems have limited resources, such as battery life, network bandwidth, storage capacity, and processor performance. These restrictions may be alleviated by computation offloading: sending heavy computation to resourceful servers and receiving the results from these servers. Many issues related to offloading have been investigated in the past decade[23].

Dinh et. Al., MCC integrates the cloud computing into the mobile environment and overcomes obstacles related to the performance (e.g., battery life, storage, and bandwidth), environment (e.g., heterogeneity, scalability, and availability), and security (e.g., reliability and privacy) discussed in mobile computing. This paper gives a survey of MCC, which helps general readers have an overview of the MCC including the definition, architecture, and applications[24]. The issues, existing solutions, and approaches are presented.

6. CONTRIBUTION OF EACH TEAM MEMBER

This literature survey has been conducted by two team members, Nishtha and Ravneet. Both members enthusiastically completed all the assigned activities throughout the process. We selected 9 conferences to gather the studies relevant to optimizing the energy consumption in mobile apps. We divided the conferences to search for relevant works in such a manner that Nishtha covered 4 conferences (ICSE, ASE, ISSTA, FSE) and Ravneet surveyed the remaining 5 conferences (USENIX Technical Conference,

MobiCom, MobiSys, OSDI, NDSS). All the relevant material for the literature review was identified from the list of conferences mentioned in our approach.

From conference survey, we together shortlisted 23 workshop papers and 49 research papers. We dropped workshop papers and each of us listed the reason for exclusion in a spreadsheet. We finalized a total of 24 relevant papers and together we categorized each paper into sections. Total of 15 papers of the empirical and functional category was finalized for the scope of this literature review.

Nishtha read 8 research papers out of 15 papers and drafted their summaries. Ravneet read rest of the 7 research papers and drafted their summaries. Both the members categorized the summarized papers based on the criteria “what is being optimized?”. All the categories have techniques listed on each related paper and the entire information has been structured in the form of this literature review by both the team members.

REFERENCES

[1] A. Boxall, "The number of smartphone users in the world is expected to reach a giant 6.1 billion by 2020", Digital Trends, 2015. [Online]. Available: https://www.digitaltrends.com/mobile/smartphone-users-number-6-1-billion-by-2020/.

[2] G. Metri, A. Agrawal, R. Peri and W. Shi, "What is eating up battery life on my SmartPhone: A case study", 2012. [Online]. Available: https://pdfs.semanticscholar.org/ 5e38/8d66d057d2a08736bcbad0 ea4bf7436d6ddc.pdf.

[3] D. Li, Y. Lyu, J. Gui and W. G. J. Halfond, "Automated Energy Optimization of HTTP Requests for Mobile Applications", Www-bcf.usc.edu, 2016. [Online]. Available: http://www-bcf.usc.edu/~halfond /papers/ li16icse.pdf.

[4] D. li, Y. lyu, J. Gui and W. Hanfold, "Automated energy optimization of HTTP requests for mobile applications.", *Www-bcf.usc.edu*, 2018. [Online]. Available: http://www-bcf.usc.edu/~halfond/ papers/li16icse.pdf.

[5] M. Hoque, M. Siekkinen, and J. Nurminen, "Using crowd-sourced viewing statistics to save energy in wireless video streaming", *Users.aalto.fi*, 2013. [Online]. Available: https://users.aalto.fi/~siekkine/pub/ MC042-hoque.pdf.

[6] D. Bui, Y. Liu, H. Kim, I. Shin and F. Zhao, "Rethinking Energy-Performance Trade-Off in Mobile Web Page Loading", *Cseweb.ucsd.edu*, 2015. [Online]. Available: http://cseweb.ucsd.edu/~schulman/ class/cse291_w17/docs/bui_webenergy.pdf.

[7] Li, A. Tran and W. Halfond, "Making web applications more energy efficient for OLED smartphones", *Www-bcf.usc.edu*, 2014. [Online]. Available: http://www-bcf.usc.edu/~halfond/papers/li14icse.pdf.

[8] M. Vasquez, C. Cardenas, G. Bavota, R. oliveto, M. Penta and D. Poshyvanyk, "GEMMA: Multi-objective Optimization of Energy Consumption of GUIs in Android Apps", 2017. [Online]. Available: https://dl.acm.org/citation.cfm?id=3098348.

[9] M. Vasquez, C. Cardenas, G. Bavota, R. oliveto, M. Penta and D. Poshyvanyk, " Optimizing energy consumption of GUIs in Android apps: a multi-objective approach.", 2017. [Online]. Available: http://www.cs.wm.edu/~denys/pubs/FSE15-GEMMA-CRC.pdf.

[10] L. Zhang, P. Pathak, M. Wu, Y. Zhao and P. Mohapatra, "AccelWord: Energy Efficient Hotword Detection through Accelerometer", *Spirit.cs.ucdavis.edu*, 2015. [Online]. Available: http://spirit.cs. ucdavis.edu/pubs/conf/li-mobisys15.pdf.

[11] M. Martins, J. Cappos and R. Fonseca, "Selectively Taming Background Android Apps to Improve Battery Lifetime.", *Cs.brown.edu*, 2015. [Online]. Available: http://cs.brown.edu/~rfonseca/pubs/ martins15tamer.pdf.

[12] X. Chen, A. Jindal, N. Ding, Y. Hu, M. Gupta and R. Vannithamby, "Smartphone Background Activities in the Wild: Origin, Energy Drain, and Optimization", *Sigmobile.org*, 2015. [Online]. Available: https://www.sigmobile.org/mobicom/2015/papers/p40-chenA.pdf.

[13] J. Gui, S. Mcilroy, M. Nagappan and W. Hanfold, "Truth in Advertising: The Hidden Cost of Mobile Ads for Software Developers", *Www-bcf.usc.edu*, 2015. [Online]. Available: http://www-bcf.usc.edu/~halfond /papers/gui15icse.pdf.

[14] C. Guo, J. Zhang, J. Yan, Z. Zhang and Y. Zhang, "Characterizing and detecting resource leaks in Android applications", *Lilicoding.github.io*, 2013. [Online]. Available: http://lilicoding.github.io /SA3Repo/papers /2013_guo2013characterizing.pdf.

[15] Banerjee, L. Chong, S. Chattopadhyay and A. Roychoudhury, "Detecting energy bugs and hotspots in mobile apps", *Comp.nus.edu.sg*, 2014. [Online]. Available: https://www.comp.nus.edu.sg/~abhik/pdf/ fse14.pdf.

[16] J. Huang, Y. Aafer, D. Perry, X. Zhang and C. Tian, "UI driven Android application reduction", *Cs.purdue.edu*, 2017. [Online]. Available: https://www.cs.purdue.edu/homes/huang427/ docs/ase17_TOFU.pdf.

[17] F. Xu, Y. Liu, T. Moscibroda, R. Chandra, L. Jin, Y. Zhang and Q. Li, "Optimizing background email sync on smartphones", 2013. [Online]. Available: https://dl.acm.org/citation.cfm?id=2464444.

[18] Y. Liu, C. Xu and S. Cheung, "Characterizing and detecting performance bugs for smartphone applications.", *Sccpu2.cse.ust.hk*, 2014. [Online]. Available: http://sccpu2.cse.ust.hk/andrewust/files/ ICSE2014.pdf.

[19] O. Devices, J. Shinar and S. York, "Organic Light-Emitting Devices - A Survey | Joseph Shinar | Springer", Springer.com. [Online]. Available: http://www.springer.com/gp/book/9780387953434.

[20] R. Ahmad, A. Gani, S. Hamid, F. Xia and M. Shiraz, "A Review on mobile application energy profiling", Thealphalab.org, 2015. [Online]. Available: http://thealphalab.org/papers/AReviewonMobileApplicationEnergyProfilingTaxonomyState-of-the-artandOpenResearchIssues.pdf. [Accessed: 16- Apr- 2018].

[21] G. Perrucci, F. Fitzek and J. Widmer, "Survey on Energy Consumption Entities on the Smartphone Platform", Pdfs.semanticscholar.org, 2011. [Online]. Available: https://pdfs.semanticscholar.org/6b67/a2eb179cad467d4433c153a4b83fdca6cee8.pdf. [Accessed: 16- Apr- 2018].

[22] J. Zhang, D. Wu, S. Ci, H. Wang and A. Katsaggelos, "Power-Aware Mobile Multimedia: a Survey (Invited Paper) - Semantic Scholar", Semanticscholar.org, 2009. [Online]. Available: https://www.semanticscholar.org/paper/Power-Aware-Mobile-Multimedia%3A-a-Survey-(Invited-Zhang-Wu/4159773c9fe0575b7b64c324cc4cb22f14dbd16f. [Accessed: 16- Apr- 2018].

[23] K. Kumar, J. Liu, Y. Lu and B. Bhargava, "A Survey of Computational Offloading in Mobile Cloud Computing - IEEE Conference Publication", Ieeexplore.ieee.org, 2013. [Online]. Available: https://ieeexplore.ieee.org/document/7474411/. [Accessed: 16- Apr- 2018].

[24] G. M. R. and K. Srinivas, "A survey of mobile cloud computing: architecture, applications, and approaches", Ijsret.org, 2014. [Online]. Available: http://www.ijsret.org/pdf/120797.pdf. [Accessed: 16- Apr- 2018].

APPENDIX A

S.No.	Conference	Category	Paper
1.	**ICSE(2016)**	NCE	Automated energy optimization of HTTP requests for mobile applications.
2.	**MobiCom(2013)**	NCE	Using crowd-sourced viewing statistics to save energy in wireless video streaming
3.	**MobiCom(2015)**	NCE	Rethinking Energy-Performance Trade-Off in Mobile Web Page Loading
4.	**ICSE(2014)**	GUI	Making web applications more energy efficient for OLED smartphones.
5.	**ICSE(2017)**	GUI	GEMMA: multi-objective optimization of energy consumption of GUIs in Android apps
6.	**FSE(2015)**	GUI	Optimizing energy consumption of GUIs in Android apps: a multi-objective approach.
7.	**MobiSys(2015)**	GUI	AccelWord: Energy Efficient Hotword Detection through Accelerometer
8.	**USENIX Technical C.(2015)**	BA	Selectively Taming Background Android Apps to Improve Battery Lifetime.
9.	**MobiCom(2015)**	BA	Smartphone Background Activities in the Wild: Origin, Energy Drain, and Optimization
10.	**ICSE(2015)**	DC	Truth in Advertising: The Hidden Cost of Mobile Ads for Software Developers
11.	**ASE(2013)**	DC	Characterizing and detecting resource leaks in Android applications.
12.	**FSE(2014)**	DC	Detecting energy bugs and hotspots in mobile apps
13.	**ASE(2017)**	DC	UI driven Android application reduction
14.	**MobiSys(2013)**	OV	Optimizing background email sync on smartphones
15.	**ICSE(2014)**	discussion	Characterizing and detecting performance bugs for smartphone applications.

NCE	Network communication energy
GUI	GUI
BA	Background activities
Dc	Device components(memory/sensors/CPU/GPS/WIFI)
OV	OVERLAPPED

APPENDIX B

Relevant Workshop Papers

S.No	Conference	Paper
1.	ICSE(2017)	Locating energy hotspots in source code
2.	ICSE(2017)	Assisting non-specialist developers to build energy-efficient software
3.	ICSE(2017)	Advancing energy testing of mobile applications
4.	ICSE(2015)	Energy-Aware Performance Evaluation of Android Custom Kernels.
5.	ICSE(2015)	EcoDroid: An Approach for Energy-Based Ranking of Android Apps
6.	ICSE(2015)	Energy Consumption Analysis of Image Encoding and Decoding Algorithms
7.	ICSE(2014)	Energy aspects: modularizing energy-aware applications (Workshop)
8.	ICSE(2014)	Can execution time describe accurately the energy consumption of mobile apps? an experiment in Android.
9.	ICSE(2014)	Green mining: energy consumption of advertisement blocking methods
10.	ICSE(2014)	An investigation into energy-saving programming practices for Android smartphone app development
11.	ICSE(2013)	A method for characterizing energy consumption in Android smartphones
12.	FSE(2015)	Improving energy consumption in Android apps.
13.	Mobicom (2013)	Characterize energy impact of concurrent network-intensive applications on mobile platforms
14.	Mobicom (2013)	Bundling frames to save energy while streaming video from LTE mobile device
15.	Mobicom (2013)	Slicing the battery pie: fair and efficient energy usage in device-to-device communication via role switching

Irrelevant Workshop Papers

S.No.	Conference	Paper
1.	**FSE(2015)**	Optimizing energy of HTTP requests in Android applications.
2.	**FSE(2015)**	Optimizing display energy consumption for hybrid Android apps (invited talk)
3.	**FSE(2014)**	Detecting energy bugs and hotspots in mobile apps
4.	**FSE(2014)**	Energy-aware design patterns for mobile application development (invited talk).
5.	**OSDI(2014)**	All Opportunities Are Not Equal: Enabling Energy Efficient App Syncs In Diverse Networks.
6.	**OSDI(2014)**	A Case for Battery Charging-Aware Power Management and Deferrable Task Scheduling in Smartphones
7.	**OSDI(2014)**	Mobile GPU Power Consumption Reduction via Dynamic Resolution and Frame Rate Scaling
8.	**OSDI(2014)**	FingerShadow: An OLED Power Optimization Based on Smartphone Touch Interactions

APPENDIX C

S.No.	Conference	Paper
1.	ASE(2016)	Battery-aware transformations in mobile applications.
2.	ASE(2015)	Recommending API Usages for Mobile Apps with Hidden Markov Model
3.	ISSTA(2017)	An actionable performance profiler for optimizing the order of evaluations.
4.	USENIX Technical C.(2015)	LPD: Low Power Display Mechanism for Mobile and Wearable Devices
5.	USENIX Technical C.(2015)	Memory-Centric Data Storage for Mobile Systems
6.	USENIX Technical C.(2016)	Energy Discounted Computing on Multicore Smartphones
7.	MobiCom(2013)	Coordinating cellular background transfers using loadsense
8.	MobiCom(2013)	Power and thermal challenges in mobile devices
9.	MobiCom(2014)	Enfold: downclocking OFDM in WiFi
10.	MobiCom(2014)	Rethink energy accounting with cooperative game theory
11.	MobiCom(2015)	Optimizing Smartphone Power Consumption through Dynamic Resolution Scaling
12.	MobiCom(2015)	Poster: Extremely Parallel Resource Pre-Fetching for Energy Optimized Mobile Web Browsing
13.	MobiCom(2016)	RnB: rate and brightness adaptation for rate-distortion-energy tradeoff in HTTP adaptive streaming over mobile devices
14.	MobiCom(2017)	RAVEN: Perception-aware Optimization of Power Consumption for Mobile Games
15.	MobiSys(2013)	Energy characterization and optimization of image sensing toward continuous mobile vision
16.	MobiSys(2013)	Spartacus: spatially-aware interaction for mobile devices through energy-efficient audio sensing
17.	MobiSys(2013)	ViRi: view it right
18.	MobiSys(2013)	Auditeur: a mobile-cloud service platform for acoustic event detection on smartphones
19.	MobiSys(2015)	Starfish: Efficient Concurrency Support for Computer Vision Applications
20.	MobiSys(2015)	Energy Efficient WiFi Display
21.	MobiSys(2015)	Energy Efficient and Fair Management of Sensing Applications on Heterogeneous Resource Mobile Devices
22.	MobiSys(2016)	MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints
23.	MobiSys(2017)	DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models using Wearable Commodity Hardware
24.	MobiSys(2017)	Enabling Cross-ISA Offloading for COTS Binaries
25.	NetGames(2012)	Managing power for closed-source android os games by lightweight graphics instrumentation

26.	**MobiCASE(2015)**	Jouler: A Policy Framework Enabling Effective and Flexible Smartphone Energy Management.