International Journal of Computer Applications (0975 – 8887)
Volume 119 – No.18, June 2015

Optimization in Power Usage of Smartphones

Balaji A. Naik
CSE Department
SGGSIE&T, Nanded-431606

R.K. Chavan
CSE Department
SGGSIE&T, Nanded-431606

ABSTRACT
The demand for smartphones and mobile based applications is growing very fast since past few years. Thousands of applications on Google Play store received millions of downloads. The growing smartphone functionalities have increased its energy requirements. The applications provide amazing features and rich user interfaces, make use of hi-tech sensors leading to high power utilization. Many such application contains various kinds of power bugs which leads to unnecessary processes running in the system. There is large scope to optimize power utilization in smartphones. This paper identifies various components in smartphones that utilize power causing unnecessary power wastage in the system. It highlights various subsystems proposed by researchers in order to optimize power consumption in smartphones.

General Terms
Mobile computing, Smartphone Energy optimization.

Keywords
Smartphone, Applications, Power usage, Energy bugs, Optimization.

1. INTRODUCTION
There has been an evolutionary change witnessed since past few years in the mobile devices. Recent smartphones utilize powerful hardware and sensors which provide greater facilities to the users. Development in smartphone technology increased the energy requirement of these devices. Smartphones are battery driven to allow mobility to the user. Many users would be satisfied if their smartphones simply lasted for many days on a single charge. Unfortunately, battery needs to be recharged several times a day under normal usage. Power utilization is limiting the development of smartphones as the improvement in battery capacity is moderate compared to the increase in the complexity due to new hardware and services [5]. Power consumption has always been an issue in smartphones and applications written by amateur programmers are making it even more difficult. As batteries can store fixed amount of power, the operational time a user can use its phone within one charging cycle is limited. The operational time is one of the important factors for consumers when they buy a new phone, therefore, the smartphone industry is very keen in finding solutions to extend the operational time.
Battery lifetime can be extended manually by managing hardware components such as GPS, 3G, Wi-Fi, Bluetooth, turning them off when they are not in use. However, this method is frustrating as users struggle to remember when to have different components on/off or they forget to enable a device they need.
Optimizing power utilization has been investigated at various levels such as system, circuit architectures, processors, memories, displays, wireless subsystems, and software [2]. A core requirement of effective and efficient management of power is a good understanding of where and how the power is used, how much of the system's power is consumed by different parts of the system.
This study identifies the power consuming components of smartphone in section 2. Performance parameters of these components under different usage scenarios are discussed in section 3. Major causes of battery draining are enlisted in section 4. Different solutions and methodologies, proposed by various researchers are discussed in section 5. Related work and conclusion are discussed in section 6 and section 7.

2. POWER CONSUMING COMPONENTS OF SMARTPHONES
A study published in 2010 by Aaron Carroll and Gernot Heiser [1] measured various key points of a smartphone. For example, they measured each component, such as the GPS, all under different power modes and usage levels. The power drain of each component varies greatly on usage. For example, calls drained 834mW and GPS 143- 166mW on the average. Screen backlight amounts to between 7-8mW to 404mW depending on brightness. It should also be noted that the testing device is a 2.5G phone and 3G drains more power.
Mobile devices sensors majorly contribute to the total power utilization of the device. The most common today are the Accelerometer, Camera, GPS, Compass, Gyroscope, Gravitation sensors, Light sensor, Proximity sensors, Pressure sensors etc. Mobile hardware such as Network Module, Flash Card, RAM, Processor, Audio and Video Codecs are also consumes lot of battery power. Figure 1 shows the major components of the smartphone that utilizes battery power. Based on the study by Carroll and Heiser [1] the power utilizations of smartphone components are discussed below.

2.1 Display
The power consumed by the display backlight over the range of an integer value between 1 and 255. This is controlled by power management module. A typical Brightness control UI gave the value between 30 and 255. The minimum backlight power is approximately 7.8mW and the maximum 414mW. The backlight consumes negligible power when disabled [1].

2.2 CPU and RAM
CPU and RAM uses a lot of power in overall system approximately around 15% [1] and it increases with usage. It is core part and needs to be awaken at all the time as long as the phone is switched on. Certain processes such as video streaming, internet browsing, CPU utilization rises it starts consuming more battery. RAM& SD Card Both are memory units and uses around 7-10% each [27] of the total battery power while performing memory read/write operations [29].

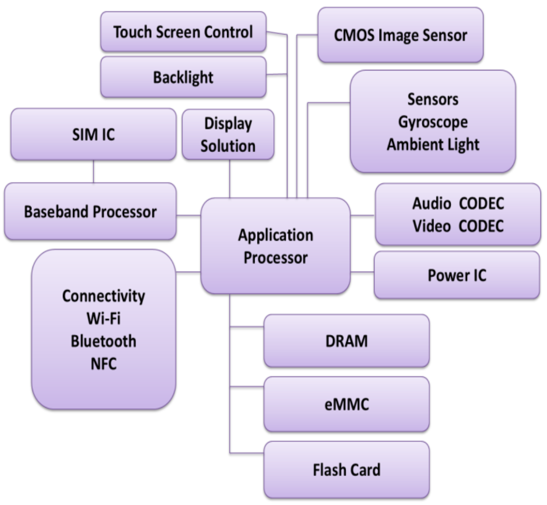

Figure 1: Power consuming components of Smartphone

2.3 Network

Wi-Fi and GPRS are biggest contributor in utilization of power. Wi-Fi showed a throughput of 660.1 ± 36.8 KiB/s, and GPRS 3.8 ± 1.0 KiB/s. The increased CPU and RAM power for Wi-Fi reflects the cost of processing data with higher throughput. The effect of signal strength on power resulted in an increase of GSM power of30%, but no effect on throughput.

2.4 GPS

The power consumed by the GPS module in three situations as shown below in Table 1 using only the internal antenna, with an external active antenna attached, and when idle [28] [1].

Table 1: GPS Energy Consumption

State	Power(mW)
Enabled (Internal Antenna)	143.1 ± 0.05%
Enabled (External Antenna)	166.1 ± 0.04%
Disabled	0.0

2.5 Bluetooth

To measure Bluetooth power utilization, the audio output to a Bluetooth stereo headset was played. The power difference between this and the baseline audio benchmark should yield the utilization of the Bluetooth module. The headset was placed approximately 30 cm from the phone, and 10m in the far benchmark as shown in Table 2.

Table 2: Bluetooth power under the audio benchmark

Benchmark	Power (mW)	
	Total	Bluetooth
Audio baseline	459.7	-
Bluetooth (near)	495.7	36.0
Bluetooth (far)	504.7	44.9

The major components are GSM module and display, LCD, Touchscreen, Graphics, Accelerator, and the backlight. The GSM module consumes a great deal of both static and dynamic energy. During phone call, GSM consumes 800mW on average. Dimming the backlight during a call saves up to 40% power even with the large GSM. The RAM, audio and flash subsystems showed the lowest power utilization. Video playing, is one of most data-intensive uses of mobile devices.

Table 3: Overall Power utilization for different parts of the mobile phone [6] [7].

Component	Action	Power utilization[mW]
CPU usage	50%	462
Wi-Fi	In connection	868
	Idle	58
	Downloading at 4.5Mbps	1450
Display	Black background 20%intensity	63.0
	White background 100%	527.05
Memory	Saving 1 Mb on drive	587.7
Voice	Making a voice call (3G)	1265.7
Video	Making a video call (3G)	2210
Bluetooth	BT on	15
	BT sending	432

3. USAGE SCENARIOS

3.1 Audio playback

The audio subsystem typically consumes 33.1mW. Approximately 58% of this energy is consumed by the codec, with the remaining 42% used by amplifier. Overall, the audio subsystem accounts for less than 12% of energy consumed. The additional energy consumed in the high-volume benchmark is less than 1mW compared with the low-volume case [1].

3.2 Video playback

A video file of 5 minute was recorded without audio and played on the same platform to measure the power usage. The brightness levels with backlight power on were 30, 105, 180 and 255. GSM power was included in the measurements. The CPU is biggest single contributor of power, the display subsystems still account for 38% of aggregate power, up to 68% with maximum backlight brightness. Negligible power requires to load the video from the SD card.

3.3 Phone call

The measurement of power utilization of a GSM phone call includes loading the dialer application, dialing a number, and making a 57-second call. Thus the time spent in the call was approximately 40 seconds, assuming a 7-second connection time. The total benchmark runs for 77 seconds. GSM power clearly dominates in this benchmark at 832:4 ± 99:0 mW. Android disables the backlight during the call.

3.4 Web Browsing

The power utilization for web-browsing workload using both GPRS and Wi-Fi connections. The benchmark ran for a total of 490 seconds which consisted of loading the browser application, selecting a web site and browsing several pages.

Here the BBC News website used. After every run, the browser cache was cleared. GPRS consumes more power than Wi-Fi by a factor of 2.5. Other components do not display any difference between the two benchmarks.
Overall, the contribution of different components to system power utilization under Different Usage Scenarios is as shown in Figure 2. In all usage scenarios, except GSM phone call, static power accounts for 50% of the total. If the backlight is included, this figure rises. This concludes that the most effective power management method on mobile is to shut down unused components and disable their power supplies.

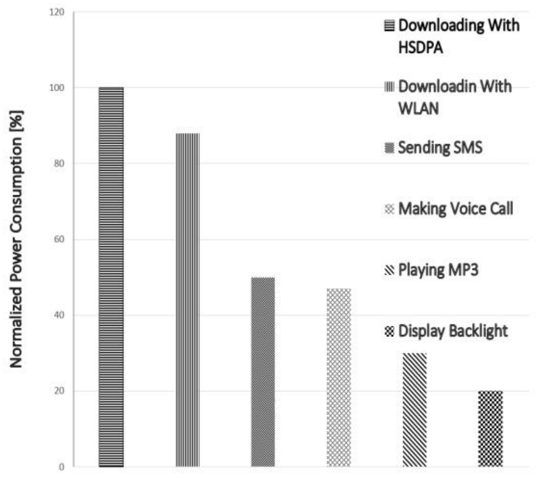

Figure 2: Energy Consumption on a Typical Smartphone

4. CAUSES OF BATTERY DRAINING

There are many aspects to look at the problem of high energy utilization in smartphones. Some of the major causes of battery draining are discussed below.

4.1 Missing sensor Deactivation Bugs

To use a sensor, an application needs to register a listener with OS, and specify a sensing rate [9].When use of sensors is finished, its listener should be unregistered in time. Forgetting to unregister sensor listeners can cause unnecessary sensing operations.

4.2 Wake lock Registration Bugs

To keep a smartphone awake for operations, Applications need to acquire a wake lock from OS and specify a wake level. A full wake lock can keep a CPU awake and its screen at full brightness. Acquired wake lock must be released as soon as the computation finishes execution. Forgetting to unregister wake locks can quickly drain the battery [10].

4.3 Sensory Data Underutilization

Sensory data are acquired at the cost of battery power. These data must be effectively used by applications to produce benefits to phone user. When an application's program logic becomes complex, sensory data may be "underutilized" in certain executions. In such executions, the power cost for acquiring sensory data may outweigh the actual usages of these data.

4.4 No-Sleep power bugs in smartphone applications

No-sleep bugs are power bugs resulting from miss-handling power control APIs in an app, which keeps smartphone components staying on for an unnecessarily long period of time. No-sleep bugs form one important category of the family of smartphone power bugs as errors in the mobile system that causes an unexpectedly high power utilization by the system as a whole.

4.5 Signal Dead Spots

Smartphone is always scanning for a signal and scanning is a power consuming task. A major source of smartphone power utilization is accessing the Internet through 3G or Wi-Fi. But wireless channel can be noisy and signal strength can be low or poor. The poor signal strength makes mobile try harder to establish connection with the network and may leads to several retransmissions [11]. Hence low signal strength or signal dead spots can significantly inflate the actual power.

4.6 Multimedia streaming

Decoding and Display are often responsible for a large part of power utilization, wireless interfaces can equally deplete the same amount of power when running audio or video streaming applications in mobiles. A typical Wi-Fi interface can use three times the power required to decode audio or video [15], whereas 3G interface requires five times the audio decoding power [12].

4.7 Web browsing

The current smartphone internet browser wastes a lot of power when downloading webpages. Wireless radio interface consumes a large amount of power even in no network traffic. Radio interface keeps connection established for further communication. It keeps radio interface to be always on and the radio resource cannot be released.

4.8 Heavy Computation programming

Large computing processes can drain smartphone battery quickly. Executing such operations increases CPU utilization to large extent. Instead Remote execution to reduce the power needs of mobile devices. Applications can take advantage of the resource-rich infrastructure by delegating code execution to remote servers. For example Firefox OS available in recent mobiles executes some of the heavy computing processes on remote severs.

4.9 Abnormal Battery Draining

ABD refers to abnormally draining of a smartphone battery that is not caused by normal usage. This power loss occur due to some triggering events or overuse of some application or hardware failure. ABD can occur while installing certain application or upgrading the previously installed one. It may also occur due to some misbehaving in the hardware while rebooting the system. Many other operations can trigger ABD such as downloading some malicious content, overusing an application etc.

4.10 Other causes for Power wastage

Power wastage can occur due to many other reasons such as keeping apps running in the background, improper charging, keeping Wi-Fi, GPS and mobile data on, keeping brightness at high level, keeping Bluetooth on, Overuse for gaming and high resolution video playbacks etc.

5. OPTIMIZATION TECHNIQUES

5.1 Diagnosing Power Bugs in Applications

Yepang Liu, Chang Xu and S.C. Cheung [17] have proposed Green-Droid: "An automated method to diagnosing power

problems in Android applications". Although the root causes of power problems can vary with different applications, many of them (over 60 percent) are closely related to two types of problems:
1. Missing sensor or wake lock deactivation.
2. Sensory data underutilization.

5.1.1 Missing Sensor or Wake Lock Deactivation
A full wake lock keeps a phone's CPU awake and its screen on at full brightness [10]. The acquired wake lock must be released as soon as the operation completes. Forgetting to unregister wake locks in time can quickly drain a phone's battery. To detect missing sensor and wake execution of an application following two policies need to be implemented:
Sensor Management Policy: A sensor listener once registered, must be unregistered eventually before the application component that registered is destroyed.
Wake Lock Management Policy: A wake lock once acquired must be released eventually before the application component that acquired is destroyed.

5.1.2 Sensory Data Underutilization
The above method of Automated Diagnosis of applications with Green-droid identifies smartphone applications that collects sensory data. These data are obtained at high power cost, so it should be utilized effectively by applications. This method addressed two issues. One is, sensory data, once received by an app, would be transformed to different forms to be used by all components of the applications. This allows sensory data usage to be tracked continuously. It tracks data usage throughout the lifetime of an application by following techniques:
Preparing and Tainting Sensory Data: The sample sensory is fed to the application under analysis after each event handler's call. Object references for each sensory datum are initialized with a unique taint mark before the datum is fed to the application.
Propagating Taint Marks: Tainting policy comprises 12rules of taint propagation. Which handles the taint propagations along data dependencies. Taint propagation terminates when the application under analysis finishes its handling of sensor event.
Analyzing Sensory Data Utilization: The program data with taint marks associated with sensory data, analysis is done on how sensory data are used in an Android application and whether the uses are effective with respect to power cost.
Architectural diagram of Working of Green-Droid is shown in Figure 3.

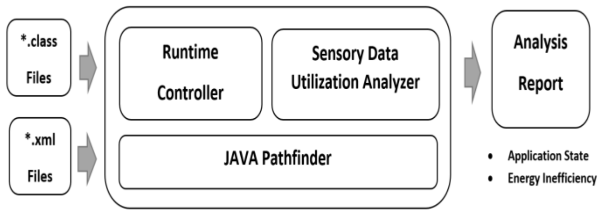

Figure 3: Green-Droid Architecture Overview

This method explores an application's state space by executing the application using Java Path Finder (JPF). It monitors sensor and wake lock operations to detect missing deactivation of sensors and wake locks. It also tracks usage of sensory data and their utilization by the application using data utilization metric. This approach can generate detailed reports with information to assist developers in detecting power problems.

5.2 Detecting No-Sleep Power Bugs in Smartphone Apps
A study proposed by Pathak & Jindal [18] in 2012 is one more attempt towards detecting the problems in applications. Study is based on no-sleep power bugs in real world apps and services. A "No-Sleep Bug" is a condition where at least one component of the phone is woken up and is not put to sleep due to a bug in power control APIs in applications [23]. Components which are woken up continues to drain the battery for a long period of time. The No-Sleep Bug detector program [18] developed by Pathak and Jindal uses following approach. The bugs are collected by crawling Internet mobile forums, bug repositories, commit logs of applications.
1) Characterization study of no-sleep energy bugs in smartphone applications.
2) Automatically detect no-sleep power Bugs [22].
3) Detecting no-sleep bugs in Android apps and framework [31].

5.3 Code Offloading to Improve Battery Life
A study published in 2010 by E Cuervoy, A Balasubramanianz, Dae-ki Cho [19] called MAUI to reduce the power needs of mobile devices applications by delegating code execution to remote servers. This idea of code execution to the remote servers utilizes program partitioning, specifying what components need to be remotely executed. Another approach by J. Flinn, D. Narayanan [16] to use full process [20] or full VM migration [13, 14] in which individual applications can migrate to the infrastructure.
MAUI combines the benefits of both approaches: it maximizes power savings through code offload. Second, it uses programming to automatically identify the remote able methods and extract only the program state needed by those methods [30]. The Figure 4 provides a high-level architecture of the MAUI which consists components such as an interface to the decision engine, a proxy to handle the control and data transfer for method which are offloaded and a profiler for instrumenting the program and collecting measures of the program's power and data transfer needs.

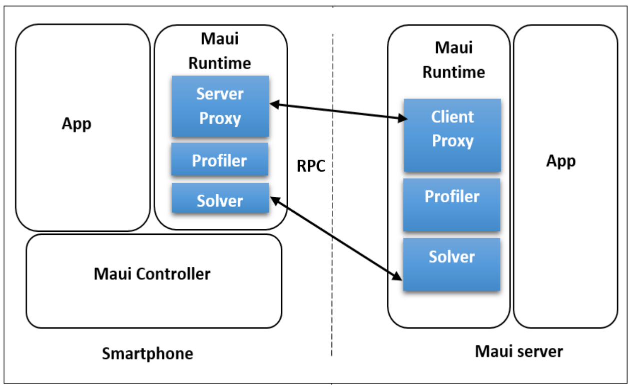

Figure 4: MAUI Architecture High level view

MAUI consists of following four components:

5.3.1 Program Partitioning
MAUI helps developers to point out remote able methods and classes that should consider offloading to a server. Code which should not be marked remote able: 1) code that implements user interface 2) Computation that interacts with I/O devices where such interaction only makes sense on the

mobile device; and 3) code that interacts with any external component that would be affected by re-execution.

5.3.2 Performing Code Offload
The proxies decides, that the method in question should be executed locally or remotely based on input from profiler, and the proxy handle both control and data transfer based on this decision. Sever side proxy performs the necessary serialization and transfers control back to the smartphone.

5.3.3 MAUI Profiler
MAUI determines whether the method invocation should run locally or remotely. The MAUI profiler measures the device characteristics at initialization time, the program and network properties.

5.3.4 MAUI Solver
The MAUI solver uses data collected by the MAUI profiler. The solver's goal is to find a program partitioning strategy that minimizes the smartphone's power utilization.

5.4 Power Optimization in Web-Browsing
Another new approach is proposed by Bo Zhao, Qiang Zheng and Guohong Cao [48] to optimize the power usage while internet browsing in 3G based smartphones. Smartphone based web browsing wastes a lot of power when downloading webpages due to the special characteristics of the 3G radio interface.
Smartphone browsers takes a long time for downloading and processing all objects in a webpage. Which consumes lots of power and decreases the network capacity. This method uses two techniques. To reorganize the computation sequence of the web browser to retrieve all data in the webpage quickly. Next, it describes how to separate objects of different types. For the execution to process HTML files, we scan them to fetch the objects referred by URLs.

5.4.1 Intermediate Display
To improve user experience, original web browsers always draw intermediate display and update it frequently when loading pages on web. By parsing 1/3 webpage content, a simplified intermediate display can be drawn. Because this display does not need CSS rules [26] or style formats, it costs little layout computation only and it can be displayed much earlier than the original web browser.
It is seen that this method reduces the data transmission time by 27%. This approach can reduce the power use by 30.9%. Most of this power saving comes from reducing the data transmission. This method is one of the best technique to optimize mobile energy consumption.

5.5 Abnormal problems of battery draining
A tool eDoctor [49] is implemented by Xiao Ma, Peng Huang, which is used to identify an abnormal problems of application. ABD refers to abnormally fast draining of a smartphone's battery that is not caused by normal usage. Major reasons of Abnormal Battery Draining are:
1) Android app problems.
2) Overuse or misuse of certain types of resources.
3) Installation of compromised smartphone application.
4) Upgrading existing app to a malicious version.
5) Changing configurations to be more power-consuming.
6) Entering a weak signal area.
eDoctor tool is to help users diagnose and resolve battery drain problems. The tool diagnosis focuses on identifying which app causes an ABD issue and which event is responsible. Based on such diagnosis result, it then suggests appropriate repair solutions [32].
This tool monitors CPU, GPS, sensors such as accelerometer and compass, wake lock, audio, Wi-Fi, and network. It also records battery utilization of every app in each recording interval. It employs profile based power model instead of expensive state-based power models [24, 25].The eDoctor records two types of events, configuration changes and maintenance events such as installation, updates. When users recognizes ABD problems, eDoctor's Diagnosis Engine, pinpoints the culprit app and the causing events and suggest suitable repair solutions.

Table 4: Comparative analysis of existing techniques

Algorithm	Type of Analysis
Green-Droid	Analysis of application source codes to find the energy bugs related to Wake-locks and sensor registrations.
No-Sleep Bug by Pathak and Jindal	Analysis of applications to find normal processes and background processes that keeps smartphone awake.
MAUI	MAUI based on delegating the heavy code executions to remote servers, in order to minimize the processing load of smartphone.
Web-Browser Optimization	While using the smartphone browsers, a light weight intermediate frame is inserted to perform page loading faster and hence minimizing energy usage.
eDoctor For ABD	ABD (Abnormal Battery Draining) caused due to some triggering of events result to significant energy wastage. eDoctor is a tool to help users diagnose and resolve battery drain problems.
Optimizing Tools Like DU Battery saver, Battery Doctor etc.	There are practical tools which directly optimize the power usage by forcing the running processes to stop and configuring the smartphone display, sensor settings for minimum power usage.

6. RELATED WORK
Researchers have worked on many different issues in the domain of smartphone energy utilization and resolved those issues using many different strategies. MAUI [30] helped offload "power-consuming" operations to resource-rich infrastructures such as remote servers. EnTracked [38] and RAPS [39] adopted various heuristics to guide an application to utilize sensors like GPS in a smart way. Little Rock [40] proposed a dedicated low energy processor for energy-consuming operations. SALSA [46] helped to select optimal data links for saving power in big data transmissions. Kim et

al [45] proposed to use power signatures on system hardware states to detect power-greedy malware.

Zhang et al [44] proposed a taint-tracking technique for the Android platform to detect power wastes due to unnecessary network communications. Power Tutor [43] uses system-level energy-utilization models to estimate the power consumed by big system components at the time of the execution of mobile applications. Different tools have been developed for detecting resource leaks [41]. For example, QVM [41] is a specialized runtime environment for detecting problems in Java programs. It monitors apps executions and checks for violations of resource safety policies.

Dynamic information flow tracking scans interesting data as they propagate in a program execution [37]. Taint- Check [36] uses Dynamic information Flow Tracking to protect commodity software from memory corruption attacks such as buffer overflows. Taint-Droid [35] prevents Android applications from leaking users' private data. It tracks such data from privacy sensitive sources, and warns users when these data leave the system. MobiBug [34] is a framework for debugging mobile applications that focuses on how to perform lightweight logging on resource-limited smartphones. MobiBug is designed as a traditional bug tracing system, to target bugs that result in app crashes. Pathak et al. [33] proposed "Eprof", a tool that performs fine-grained energy profiling by tracing system calls.

7. CONCLUSION

Optimization in power usage of applications in smartphone has become an important field for research in today's IT world. The major reasons for battery draining in smartphones are Network Data Communication such as Multimedia Streaming, GPS, Wi-Fi, and Signal Dead Spots. Usage scenarios such as high level of Backlight, high resolution Video Playbacks, Graphics Rich Gaming, and Heavy Computing Processes are the main sources power consumption. Other causes of power wastage are Application Energy Bugs, No- Sleep Bugs, Unnecessary use of Sensors and continuously running Background Processes. Wake Locks and Sensors can also quickly drain the battery if the programmers forget to unregister it in time. This study highlighted the problems and the solutions for optimization of energy consumption in smartphones.

8. REFERENCES

[1] A. Carroll and G. Heiser, "An analysis of power utilization in a smartphone", in Proceedings of the 2010 USENIX conference on USENIX annual technical conference, USENIXATC10, (Berkeley, CA, USA), pp. 21-21, USENIX Association, 2010.

[2] Olsen, C.M. Narayanaswami, "Power-Nap: An efficient power management scheme for mobile devices", IEEE Trans. On Mobile Computing, Vol. 5, No. 7, pp. 816-828. 2006.

[3] "Nanowire battery can hold 10 times the charge of existing lithium-ion battery", Stanford technical report, Stanford, 2007.

[4] Q. Naing, V. Hoffer, J. A. Weber, D. J.Kuo, A. D. Donelan, J. M. Li, "Biomechanical power harvesting: Generating electricity during walking with minimal user effort", Science, Volume 319, Issue 5864:807, 2008.

[5] Findlay Shearer, "Power management in mobile devices", chapter Hierarchical View of Power Conservation, pages 32-75. Newnes, 2008.

[6] G.P. Perrucci, F.H.P Fitzek, J. Widmer, "Power Utilization Entities on the Smartphone Platform".

[7] www.forum.nokia.com/devices/n95.

[8] Sean Maloney, Ivan Boci, "Survey on Techniques for Efficient power utilization in Mobile Architectures", March 16th, 2012.

[9] Android Sensor Management. (2013). [Online]. Available:http://developer.android.com/reference/android/hardware/SensorManager.html

[10] Android power management. (2013). [Online]. Available:http://developer.android.com/reference/android/os/PowerManager.html

[11] Ning Ding, Daniel Wagner, Xiaomeng Chen, "Characterizing and Modelling the Impact of Wireless Signal Strength on Smartphone Battery Drain".

[12] Mohammad A. Hoque, MattiSiekkinen, and Jukka K. Nurminen, "On the power efficiency of proxy-based traffic shaping for mobile audio streaming", in Consumer Communications and Networking Conference.

[13] B.-G. Chun and P. Maniatis, "Augmented Smartphone Applications through Clone Cloud Execution", in Proc. Of the 8th Workshop on Hot Topics in Operating Systems (HotOS), Monte Verita, Switzerland, May 2009.

[14] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-based Cloudlets in Mobile Computing", IEEE Pervasive Computing, 8(4), 2009.

[15] Surendar Chandra, Amin Vahdat, "Application-specific network management for power-aware streaming of popular multimedia formats", in Proc. General Track of the annual conference on USENIX Annual Technical Conference, pages 329–342. USENIX, 2002.

[16] J. Flinn, D. Narayanan, and M. Satyanarayanan, "Self-Tuned Remote Execution for Pervasive Computing", In Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS), Schloss Elmau, Germany, May 2001.

[17] Yepang Liu, Chang Xu, S.C. Cheung, "Green-Droid: Automated Diagnosis of Energy Efficiency for Smartphone Applications", in proc. IEEE Transactions on Software Engineering. Vol. 40 Sept 2014.

[18] A Pathak, A. Jindal, "Characterizing and Detecting No-Sleep Energy Bugs in Smartphone Apps", MobiSys'12, June 25–29, 2012.

[19] Eduardo Cuervoy, Aruna Balasubramanianz, Dae-ki Cho, "MAUI: Making Smartphones Last Longer with Code Offload", in MobiSys'10, San Francisco, California, June 15–18, 2010.

[20] S. Osman, D. Subhraveti, G. Su, and J. Nieh, "The Design and Implementation of Zap: A System for Migrating Computing Environments", in Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI), 2002.

[21] J. Liu and L. Zhong, "Micro power management of active 802.11interfaces", in Proc. ACM MobiSys, 2008.

[22] A. Aho, M. Lam, R. Sethi, and J. Ullman, "Compilers: principles, techniques, and tools", Pearson/Addison Wesley.

[23] A. Pathak, Y. C. Hu, and M. Zhang, "Bootstrapping power debugging for smartphones: A first look at power bugs in mobile devices", in Proc. of Hotnets, 2011.

[24] Shye, A., Scholbrock, B., and Memik, G., "Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures", in Proceedings of the 42nd Annual IEEE/ACM International Symposium on Micro architecture(2009),Micro 42, ACM, pp. 168–178.

[25] Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., and Yang, L., "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones", in Proc. of the eighth IEEE/ACM/IFIP International Conference on Hardware/software Code sign and System Synthesis(2010), CODES/ISSS '10, ACM, pp. 105–114.

[26] L. A. Meyerovich and R. Bodik, "Fast and parallel webpage layout," in Int'l Conf. on World Wide Web (WWW), 2010.

[27] Snowdon, D. C., Le Sueur, E., Petters, S. M., And Heiser, G. "Koala: A platform for OS-level power management", in Proceedings of the 4th EuroSys Conference, Nuremberg, Germany, Apr. 2009.

[28] U-BLOX AG. ATR0630 Data Sheet, July 2006. GPS.G4-X-06009-P2.

[29] Bircher, W. L., And John, L. K, "Analysis of dynamic power management on multi-core processors", in Proceedings of the 22nd International Conference on Supercomputing (Island of Kos, Greece, June 2008), pp. 327–338.

[30] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in Proc. Int. Conf. Mobile Syst., Appl. Serv., 2010,pp. 49–62.

[31] Abhinav Pathak, Abhilash Jindal, Y. Charlie Hu, Samuel P. Midkiff. "What is keeping my phone awake? Characterizing and Detecting No-Sleep Energy Bugs in Smartphone Apps".

[32] Xiao Ma, Peng Huang, XinxinJin, Pei Wang, Soyeon Park, Dongcai Shen. "eDoctor: Automatically Diagnosing Abnormal Battery Drain Issues on Smartphones".

[33] Pathak, A., Hu, Y. C., and Zhang, M., "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 (2012), pp. 29–42.

[34] S. Agarwal, R. Mahajan, A. Zheng, and V. Bahl, "There's an app for that, but it doesn.t work. Diagnosing mobile applications in the wild", in Hotnets, 2010.

[35] W. Enck, P. Gilbert, B. G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An information-flow tracking system for real time privacy monitoring on smartphones", in Proc. USENIX Conf. Operating Syst. Des. Impl. 2010, pp. 393–407.

[36] J. Newsome and D. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software", in Proc. ISOC Netw. Distrib. Syst. Security Symp., 2005.

[37] V. P. Kemerlis, G. Portokalidis, K. Jee, and A. D. Keromytis, "Libdft: Practical dynamic data flow tracking

For commodity systems", in Proc. ACM Conf. Virtual Exe. Env., 2012, pp. 121–132.

[38] M. B. Kjærgaard, J. Langdal, T. Godsk, and T. Toftkjær, "Entracked: Energy-efficient robust position tracking for mobile devices", in Proc. Int. Conf. Mobile Syst. Appl. Serv., 2009, pp. 221 234.

[39] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive GPS-based positioning for smartphones", in Proc. Int. Conf. Mobile Syst. Appl. Serv., 2010, pp. 299–314.

[40] B. Priyantha, D. Lymberopoulos, and J. Liu, "LittleRock: Enabling energy-efficient continuous sensing on mobile phones", in IEEE Pervasive Compute., vol. 10, no. 2, pp. 12–15, Apr.–Jun. 2011.

[41] M. Arnold, M. Vechev, and E. Yahav, "QVM: An efficient runtime for detecting defects in deployed systems", in ACM Trans. Software Eng. Methodology., vol. 21, pp. 2:1–2:35, 2011.

[42] I. Dillig, T. Dillig, E. Yahav, and S. Chandra, "The CLOSER: Automating resource management in Java", in Proc. Int. Symp. Memory Manage, 2008, pp. 1–10.

[43] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones", in Proc. Int. Conf. Hardware/Software. Codes. Syst. Synthesis, 2010, pp. 105–114.

[44] L. Zhang, M. S. Gordon, R. P. Dick, Z. M. Mao, P. Dinda, and L. Yang, "ADEL: An automatic detector of energy leaks for smartphone applications" in Proc. 8th IEEE/ACM/IFIP Int. Conf. Hardware/Software. Codes. Syst. Synthesis, 2012, pp.363–372.

[45] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants", in Proc. Int. Conf. Mobile Sys. App., Serv., 2008, pp. 239–252.

[46] M. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely, "Energy-delay tradeoffs in smartphone applications", in Proc. Int. Conf. Mobile Syst. Appl. Serv., 2010, pp. 255–270.

[47] Narendran Thiagarajan, Gaurav Aggarwal and Angela Nicoara. "Who Killed My Battery: Analyzing Mobile Browser Energy Consumption", in Proc. International World Wide Web Conference2012 – Session: Mobile Web Performance Lyon France April 16–20, 2012.

[48] Bo Zhao, Qiang Zheng, Guohong Cao, "Energy-Aware Web Browsing in 3G Based Smartphones", in Proc. IEEE 33rd International Conference on Distributed Computing Systems.2013.

[49] Xiao Ma, Peng Huang, XinxinJin, Pei Wang, "eDoctor: Automatically Diagnosing Abnormal Battery Drain Issues on Smartphones".