MASTER IN COMPUTER SCIENCE

Techniques for analyzing and optimizing the energy consumption in (mobile) applications

Roosens, Simon

Award date:
2020

Awarding institution:
University of Namur

Link to publication

UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

Techniques for analyzing and optimizing the energy consumption in (mobile) applications

Simon ROOSENS

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

Université de Namur
Faculty of Computer Science
Academic Year 2019–2020

Techniques for analyzing and optimizing the energy consumption in (mobile) applications

Simon ROOSENS

Internship mentor : /

Supervisor : ______________________ (Signed for Release Approval - Study Rules art. 40)
Wim VANHOOF

Co-supervisor : /

A thesis submitted in the partial fulfillment of the requirements
for the degree of Master of Computer Science at the Université of Namur

Acknowledgements

I wish to express my sincere appreciation to my supervisor, Doctor Win Vanhoof, for his support and guidance despite the short time at his disposal to spend on the subject. His help allowed me to carry out this work in a field that was little known to me at the time. I wish to acknowledge also the support and great love of my family, friends, colleagues and others, with whom I've had the chance to share and discuss these topics, which helped me go through this work and pushed me forward to finalize this work. Finally, I would like to express my gratitude to the Université of Namur and faculty members who allowed me to pursue this master's degree in its facilities with a high level of education that lives up to my expectations. I would also like to applause all the efforts achieved by the university and faculty staff that showed agility and responsiveness to keep the evaluation sessions despite the serious health crisis that is hitting us very hard.

Abstract

EN The smartphone revolution has significantly changed our daily lives, offering a wide range of functionalities; from the ability to order food delivery in a few seconds, to the usability of finding our way around in a new city using a map application. However, the introduction of smartphones and tablets generates new challenges such as the limited battery autonomy to power and keep all these components and applications operational. Therefore, optimal and efficient energy management of the smartphone's resources becomes critical. In this approach, it is important to assess the energy consumption of the components, and further more to suggest improvements to these in a context where software developers are running out of tools to efficiently fight energy-drains. Indeed, this field of research is still in full growth and existing solutions are either not yet ready or not standardized. This state of the art explores and analyses the smartphone ecosystem, its open problems and challenges in the domain of energy consumption. Furthermore, it evaluates the different solutions that have a promising future in the field of analysis, estimation and optimization of the energy consumption of mobile applications. This paper finally summarizes the different approaches, their achievements, their contributions to the field and the possibilities for future work.

FR La révolution des smartphones a significativement changé notre quotidien, nous proposant une large palette de fonctionnalités, de la capacité de commander une livraison de nourriture en un temps record, à la facilité de pouvoir s'orienter dans une ville inconnue en utilisant un service de géolocalisation. Cependant, l'introduction des smartphones et des tablettes a généré de nouveaux défis telle que l'autonomie encore plus limitée de la batterie. En conséquence, une gestion optimale et efficace de la consommation d'énergie des ressources des smartphones devient cruciale. Dans cette démarche, il paraît important d'être en mesure d'évaluer la consommation des composants utilisés par les smartphones, mais également de proposer des pistes d'amélioration dans un contexte où les développeurs de logiciels ne disposent pas d'outils efficaces pour affronter à armes égales les problèmes de perte d'énergie. En effet, ce domaine de recherche est en pleine expansion et les solutions existantes sont soit peu prêtes, soit pas standardisées. Cet état de l'art explore et analyse l'écosystème des smartphones, ses problématiques actuelles et ses défis dans le domaine de la consommation énérgétique. De plus, ce travail évalue les différentes solutions avec un avenir prometteur en matière d'analyse, d'estimation et d'optimisation de la consommation d'énergie des applications mobiles. Ce papier clôture en résumant les différentes approches, les avancements, les contributions dans ce domaine et les pistes pour des travaux futurs.

Keywords smartphone, battery-drain, energy-saving techniques, energy profiling, issues, power consumption estimation, power consumption modelling, power consumption optimization, power management, resources usage, energy modelling, energy-efficiency, state-of-the-art, developer energy awareness.

Contents

Chapter 1

Introduction

1.1 Context

The recent introduction of smartphones allowed us to perform a large amount of tasks which are similar to those previously performed by personal computers. The field of personal computers have faced and is still facing a lot of challenges with the intention of offering an ever more advanced user-experience. It has required a lot of work from hardware and software developers to reach the level of usability we know today. Similar to the development of personal computers, the arrival of smartphones has led to new issues. While offering small size capacity, the smartphone must integrate the latest electronic components to remain competitive and meet the performances demanded by the market. High-performance resources for mobile applications require considerable efforts to limit the battery drain and to continue to provide the users with features at fingertips. At last, resources are powered from a battery with limited autonomy, which arises the never-ending search for the perfect compromise between optimising smartphones' size and increasing the battery capacity.

1.2 Problematic

While electronic component manufacturers keep on working on techniques that reduce the energy consumption of their product or increase its energy capacity, mobile application software developers focus less on this aspect which is commonly left behind. Application energy-efficiency is however an important criteria for the overall user satisfaction. Yet software development companies have few or no tools at their disposal to accurately analyze and efficiently optimize the energy consumption of their products. External measurement equipment is expensive and requires expertise that is often beyond the scope of a programmer. Moreover, setting the testing environment and run the measurements is both complex and time consuming. As this is becoming a global problem, several companies are setting up working groups (e.g GreenIT[1]) to deal with the issue. This is part of a general approach that aims to reduce the environmental footprint of technologies. Several mobile vendor members such as IBM, Apple, Intel or Microsoft are currently contributing to this problematic's solution by providing tools and documentation ([13, 20, 1, 51, 49]). However, developers are still lacking reliable, accurate, time- and effortless tools to properly analyze the power consumption of their applications.

In this thesis, we aspire to provide answers to the following question, **how software engineering solve the application energy consumption problematic using program analysis techniques?**

1.3 Contribution

In this research work, we evaluate state-of-the-art techniques to understand the current problems faced by researchers in this field. We also compare and criticize their solutions in an attempt to address the problem.

This work extends a previous research carried out during the 2017-2018 academic year. It focused on possible approaches to accurately estimate and analyse application's energy cost. As smartphone's resources usage is the initial cause of the application's energy consumption, it became interesting to expand that scope and to include the analysis of electronic components and solutions in regards to the optimization of their power consumption using program analysis. The previous work is here updated with researches conducted over the past two years. The recent literary review proposed by P. K. D. P. and al [40] was particularly helpful in carrying out the analysis on the energy consumption of electronic components (see Smartphone Power Consumption Analysis chapter).

This thesis is divided into separate chapters related to the analysis of electronic components and their impact on energy consumption, such as the processor, memory, display, graphics unit, wireless interfaces, sensors, and others; the estimation and analysis of applications and hardware components; and techniques to optimize mobile applications and hardware interfaces for main smartphone resources. Finally, we conclude by summarizing the contributions of researchers in the various fields, their relevance and future prospects.

1.4 Terms Definition

This paper is using the terms power consumption or energy consumption which is expressed as a watt, a derived unit of power in the International System of Unit (SI). Watts is defined as 1 Joule per second and is used to refer to cumulative energy consumption over time.

[1]GreenIT [88] is an emerging discipline concerned with the optimization of software solutions with regards to their energy consumption.

The term *voltage* refers to the difference in electrical potential between two points. The voltage is expressed in units of Volt (V).

The term *current* refers to a movement of electric charges that pass through a material. The current is expressed in units of Amperes (A).

The term *frequency* refers to the number of occurrences of a repeating event per unit of time. The frequency is measured in units of Hertz (Hz).

The term *power model* or *energy modelling* refers to a process of building computer models for purpose of analysis.

The term *cycle-accurate* refers to a clock cycle accuracy. In this context, it means that a system is able to match the speed of the Central Processing Unit (CPU) which can process some of the instructions every cycle.

The term *dynamic voltage scaling* is a power management technique in computer architecture, where the voltage used in a component is increased or decreased, depending upon circumstances[2].

The term *dynamic frequency scaling* is a technique in computer architecture whereby the frequency of a microprocessor can be automatically adjusted "on the fly" depending on the actual needs, to conserve power and reduce the amount of heat generated by the chip[3].

The term *transition* refers to a computer science paradigm in the context of communication systems which describes the change of communication mechanisms, i.e., functions of a communication system, in particular, service and protocol components[4].

[2]en.wikipedia.org/wiki/Dynamic_voltage_scaling

[3]en.wikipedia.org/wiki/Dynamic_frequency_scaling

[4]en.wikipedia.org/wiki/Transition_(computer_science)

Chapter 2

Smartphone Power Consumption Analysis

A software component does not consume any energy at first glance. However, it requires components at lower layers to execute its instructions, to store information whether volatile or not, to share its data and to communicate them to the outside world, and to display this information to a user. All these operations have an energy and time cost that is caused by the time and energy required for a resource to perform them. Before being able to analyze and estimate the smartphone energy consumption of these operations, it is important to understand the main sources that demand this time and energy. This section describes and analyses the sources of energy consumption that can be found in a smartphone. We separated these energy-drain sources into 2 groups: hardware components and signalling modules.

2.1 Power Consumption of the Hardware Components

There are many electronic components in a smartphone. Some are required to perform computational operations, some to store information, some to display data to the user, and some to retrieve information about the environment around the smartphone. All these components interface with the software components and operate like an ecosystem, developing a dense network of dependencies and information exchanges. This section describes what these sources of energy consumption are and what the major reasons inducing energy-drain are. P. K. D. P and al. [40] described the different physical components found in a smartphone and the factors responsible for the overall energy consumption. Their work tries to solution the following questions about hardware component parameters [83]:

- The power consumed by the component operating at idle state[1] which is the minimum power needed to be at active state.

[1]An electronic component is described as idle when it is not being used by any program.

- The purpose of the hardware component.
- The number of power state models defined
- The number of transitions available in the power states.
- The power cost of each transition.

Table 2.1: Summary of different sources and reasons for power consumption defined by [40]. The probable ways to minimise cannot be always applied, it depends on the context of use.

Power Consumption Category	**Component**	**Probable Ways to Minimise**
Hardware Components	CPU	Voltage regulation Clock frequency regulation Minimizing switching activities Code optimization Using GPU for computing Hardware accelerators Dedicated functions
	GPU	Resolution scaling Refresh rate scaling Hardware accelerators Dedicated functions
	Memory	Avoid increasing RAM size unnecessarily Usage management
	Storage	Reduced read-write operations Temperature control
	Display	Efficient technology Resolution scaling Lowering pixel density Decreasing the refresh rate Reducing backlight
	Sensors	Restricted application permission
	Cellular Network	Network mode selection Increased signal strength Controlled handoff Operator selection
	Bluetooth	Manual starting when needed Using other modes of data sharing Introducing BLE

Signalling Modules	Hotspot	Preferring USB tethering Using Bluetooth instead of Wi-Fi Using lower frequency band
	Wi-Fi	Manual starting when need Using lower frequency band Get closer to the Wi-fi access point
	GPS	Manual starting when needed Use GPS where the satellite signal is decent Using optimized GPS application
	FM Radio	Manual starting when needed Antenna management Display standby during FM radio usage
Software	Operating System	Using smartphone-oriented kernel Using kernel-level display server
	Applications	Verified installations Managing application permissions Uninstalling unused applications Disabling auto start-up applications
	Calling	Use 2G network wherever applicable Avoid calling with poor signal strength
	Internet usage	Selection of best connection Restrict network access of applications Restrict application background data
	Gaming	Selecting thermally conductive material Effective internal cooling technology Downclocked CPU Lowering display brightness

Usage Patterns	Music playback	Standby display Activated sleep mode Controlled volume Optimised audio bit-rate
	Video playback	Avoid high-resolution videos, where applicable Go for videos with low frames per second (FPS)
	Running heavy app.	Code offloading Computating offloading
Other Misc. Factors	Inter-device comm.	Inter-device task distribution Optimized processing across devices Optimized scheduling Optimized data-transfer Energy-efficient communication protocols
	Heating	Device material selection Not charging and discharging simultaneously
	Ageing and faulty bat.	Battery replacement Using power-saving mode

CPU

The CPU is in charge of executing the software's machine instructions. It consists of a computing unit, a control unit, an input-output unit, a clock and registers. The various factors responsible for the CPU's power consumption can be categorized as follows [40]:

- **Dynamic power consumption:** A CPU contains millions of transistors[2] continuously operated to perform instructions. Each transistor is toggled from one state to another and will request the capacitors to rapidly charge and discharge. This causes energy-drain.
- **Short-circuit:** During operations, the transistors change their state by either switching to ON or OFF. However, the delay for switching state may vary from one transistor to another. When a transistor takes longer to perform a transition, it can cause a short-circuit.
- **Leaking transistor:** Transistors are semiconductors that are doped to block or pass current. Mainly due to state, size and temperature, there is a small portion of the current that is leaking out.
- **Clock frequency:** The CPU frequency clock requires power to operate, the higher the frequency, the higher the power consumption. Reason being

[2]A transistor is a semiconductor device used to amplify or switch electronic signals and electrical power. Source: en.wikipedia.org/wiki/Transistor

the number of operations requested to be performed in a given time. This induces energy consumption and energy leakage. Yet each microprocessor have an optimal power consumption point where the CPU capability can remain unaffected [57].

GPU

The Graphical Processing Unit (GPU) provides the rendering features for the display. As these functions were traditionally performed by the CPU, it is becoming more and more common to have a hardware component dedicated to graphics tasks such as video rendering, 3D rendering[3], and other image processing needed in the field of video games. The categories can be defined [40] as follows:

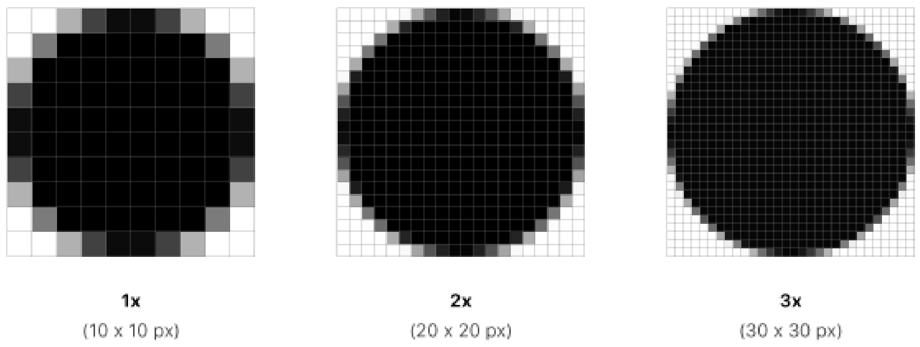


Figure 2.1: Example of pixel density, from left to right: Low (1x), Medium (2x), High (3x). blog.prototypr.io/making-sense-of-device-resolution-pixel-density

- **3D rendering:** Some rendering tasks require optimized hardware in order to efficiently compute and display polygons. The CPU does not support well the rendering tasks demanded is recent applications and games. On the opposite, the GPU manages properly such operations but still requires much energy consumption to perform them.
- **Resolution:** The resolution is the number of pixels found on a display in each dimension, width and height. This property is driven by the GPU or the CPU and has an impact directly proportional to power consumption [40].
- **Pixel Density:** Pixel density or commonly called Pixel Per Inch (PPI) defines the number of pixels forming a square (sometime pixels can be rectangular) area of one inch long (Figure 2.1). Pixel processing is driven by the GPU or the CPU. More pixels to deal with equal more power consumption required from the processing unit. A higher pixel density leads to a higher processing unit power consumption.

[3]3D rendering is the 3D computer graphics process of converting 3D models into 2D images on a computer. 3D renders may include photorealistic effects or non-photorealistic styles. Source: en.wikipedia.org/wiki/3D_rendering

- **Refresh rate:** The refresh rate is the number of times a display is updated with new images each second. The higher the refresh rate is, the smoother the displayed image is (Figure 2.2). The refresh rate defines the frequency of images processed by the graphical unit. This property is driven by the GPU or the CPU and is directly proportional to the power consumption.

Figure 2.2: Example of refresh rate effect on moving cars, from 60 Hz to 120 Hz. Higher the refresh rate is, smoother the picture will be. Picture found on www.coolblue.be

Memory

The role of memory is to store working data, mainly in a volatile manner. Allowing this data to be read and written. This component is no exception to the rule, it tends to grow in order to store an ever-increasing amount of data. A memory size-enery-cost relationship can be suggested. Researchers identified [40] three main factors that have an impact on this energy consumption:

- **Steady-state[4] power consumption:** The Random Access Memory (RAM) semiconductors have the same behavior as a capacitor[5] but store data instead of a charge. In order to keep the data stored, the RAM requires continuous power.
- **RAM capacity:** The RAM component is made out of Metal Oxide Semiconductor Field Effect Transistor (MOSFET)[6] which requires an amount of

[4]A steady-state is an equilibrium where the system is in relative stability.

[5]A capacitor is a device that stores electrical energy in an electric field. Source: en.wikipedia.org/wiki/Capacitor

[6]MOSFET is a type of transistor, also known as the metal–oxide–silicon transistor (MOS transistor, or MOS). Its ability to change conductivity with the amount of applied voltage can be used for amplifying or switching electronic signals. Source: en.wikipedia.org/wiki/MOSFET

energy to keep the steady-state. Increasing the size of the memory implies to increase the total number of MOSFETs. Thus, the energy drain for the memory is directly proportional to the size of memory.
- **Read-write power consumption:** Energy is also needed as the system is accessing the RAM. Indeed, each read/write operation requires an amount of energy to be executed. This amount depends on the technology and architecture of the RAM.

Storage

Smartphones have 2 categories of storage, internal storage which is mostly soldered on the motherboard and external storage. The technologies used have a direct impact on energy consumption. Here are the main factors:

- **Read-write power consumption:** As with any component that stores data, accessing and alternating this information requires read and write operations. The more operations are performed, the more power this component consume.
- **Temperature:** Operating temperature also plays a role in energy consumption. The controller of these storage devices can protect itself in case this temperature is too high. Its countermeasure is to reduce its frequency, which causes an increased demand for power over time to complete operations.

Display

There are various technologies available for smartphone's display. More recent Light-Emitting Diode (LED) displays are low power consumer compared to Liquid Crystal Display (LCD) displays. Similar to GPU properties, the pixel density and the refresh rate are directly proportional to power consumption. Amongst display's components, the backlight remains the biggest energy consumer. This component is required for technology displays where no other light source is possible (e.g LCD). In such case, the number of LEDs used for the backlight and their intensity is proportional to the component's power consumption.

Sensors

Sensors play an important role in the ecosystem of a smartphone. Their job is to broadcast a whole range of data to the nervous system to be analyzed, interpreted and used for many purposes. On average, there are about ten of them embedded in the device (Figure 2.3). They can be internal or external [74]. The internal sensors monitor vital signs such as battery life, central processing temperatures, status of the network, and so on. External sensors, on the other hand, can probe and capture the environment around the smartphone. Indeed, they provide a range of data such as location, video recording, audio recording, physical proximity, external temperature, external brightness, etc.. In order to

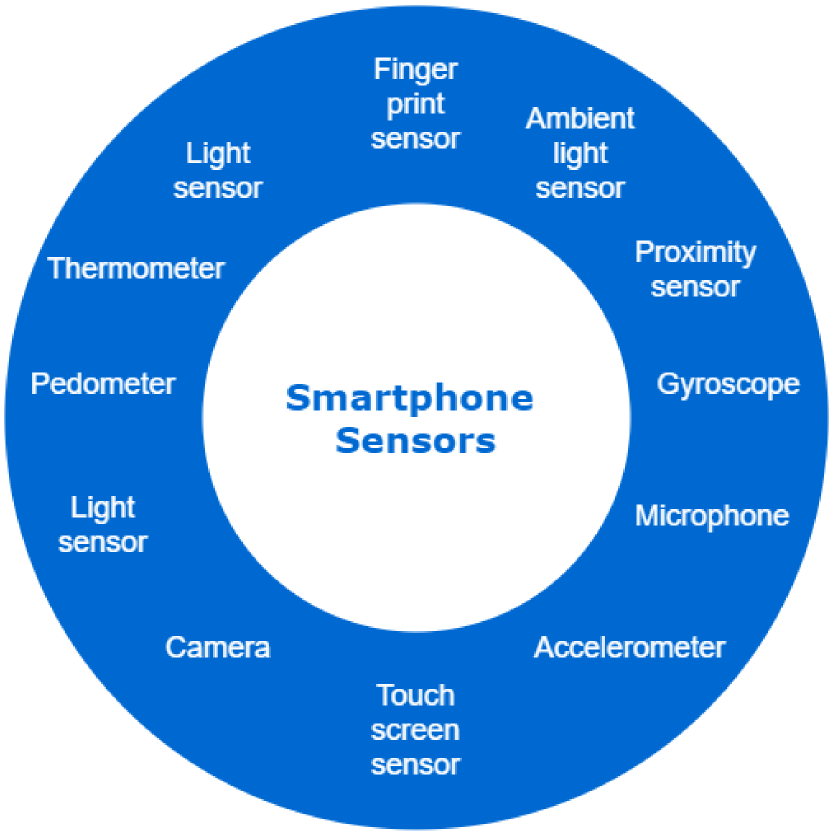


Figure 2.3: Example of sensors in a smartphone

ensure that the data used during the process of interpreting is not outdated, the sensors require to be constantly monitored, which costs a constant energy drain.

2.2 Power Consumption of the Signalling Modules

Signalling modules as well as networking modules are the most energy consuming elements in a smartphone (Figure 2.4). This is mainly because they have to transmit and receive a lot of data at high rate. These signals have to be transmitted over a greater or lesser range, which is a parameter that has a great influence on this energy cost. This chapter describes the different types of modules and the main factors influencing their energy consumption.

Cellular Network

The primary function of a phone is to make calls using the cellular network anywhere in the world. In order to ensure the highest possible call quality regardless of location, smartphones may need to amplify signals on the modules that receive and transmit data. The worst situations can lead to enormous power

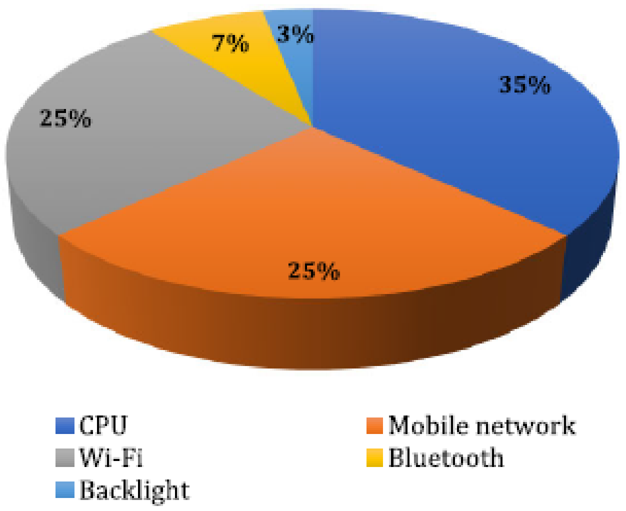


Figure 2.4: Hardware modules power consumption percentage share according to Pramanik and colleagues [40]

consumption. The main factors that cost energy are [40]: the cellular frequency[7]; the signal strength and the network operator.

Bluetooth

Bluetooth is a wireless communication standard that allows the bidirectional exchange of data over very short distances using Ultra High Frequency (UHF) radio waves on a frequency of 2,4 GHz [19]. This technology is very widely spread among smartphones where many uses are allowed. There are several versions of this standard (Table 2.2), called *Class*, all aimed at improving data transfer and energy consumption. Here are the various factors that impact the energy required to operate:

- **Constant connectivity:** Some devices (Bluetooth speaker, smart watch) paired to the smartphone via Bluetooth need to be constantly connected to maintain data flow. This need has a constant energy cost because Bluetooth transmission has to always be turned on.
- **Auto-searching devices:** Even if the smartphone is not paired with a Bluetooth device, it automatically searches for devices to connect to. This implies a consumption of this component.
- **Networking:** The proposed use of many devices is potentially not appropriate because not adapted to large data transfer [40]. The maximum distances during these transfers is also an element which can lead to a higher energy consumption.

[7]Cellular frequencies are the sets of frequency ranges within the ultra high frequency band that have been assigned for cellular-compatible mobile devices, such as mobile phones, to connect to cellular networks. Source: en.wikipedia.org/wiki/Cellular_frequencies

Wi-Fi

The Wi-Fi technology is a set of wireless communication protocols widely used in most parts of the world. A Wi-Fi network allows to connect several computer devices (computer, router, smartphone, Internet modem, etc.) within a computer network by radio waves in order to allow the transmission of data between them [93]. Here are the features that can increase energy consumption:

- **Wakeup power consumption:** Every time the Wi-Fi module is started, a wireless access process needs to be performed as defined by the 802.11 standard [50]:
  1. **Scan**: The scanning process is required to discover wireless networks in the smartphone's range. This step is performed using probes[8] that interrogate access points. The answers describe the configuration and capabilities of the wireless network managed by the access point [10]. Once a candidate wireless network has been selected, the authentication stage comes into play.
  2. **Authentication**: It is a process during which the credentials provided by the user are compared to a database on the local router or remote server. If these identifiers match, the process is completed and the user is authorized to access the network.
  3. **Association**: During the association process, the access point sends information such as SSIDs[9] and data rate to the requester. The client then deploys a probe to scan the available channels in order to associate the access point with the strongest signal. If this signal weakens, the client restarts a scan to find a better match [89].

  This waking up process is consuming power at every Wi-Fi start.
- **Connection maintenance power consumption:** Access point requires that the Wi-Fi device maintain an active connection which drains constant power.

GPS

Global Positioning System (GPS) is a hardware component used for tracking the location of the device. It exploits the satellite data to give a great precision position. Combined to applications, it can provide location services for geolocation, advertising and more. Here are the consumption details of GPS pointed by paper [40]:

[8]A probe request is a special frame sent by a client station requesting information from either a specific access point, specified by SSID, or all access points in the area, specified with the broadcast SSID. Source: www.hak5.org/episodes/haktip-23

[9]A SSID (Service Set IDentifier) serves as network name and is usually defined as a natural language label.

Bluetooth Class	Power Cons. in mW
1	100
1.5	10
2	2.5
3	1
4	0.5

Table 2.2: Power consumption of different Bluetooth classes. Higher the class is, lower the power consumption is required to operate.

- **GPS receiver:** GPS receiver is composed of an antenna tuned to satellite frequency emission, receiver-processor and an highly stable clock. A connection is required to be maintained between the smartphone and the satellites when operating.
- **Dependency:** The GPS receiver itself is only able to identify the geographic location in terms of coordinates. For example, geolocation applications need to use services such as internet map services to be able to provide a map. Moreover, the GPS can increase its accuracy by using data from cellular towers (Figure 2.5) and Wi-Fi networks. The use of multiple services has a high energy cost but for best accuracy and more.
- **Sleep mode:** When using the GPS receiver, the network between smartphone and satellites is constantly operating and data received have to be processed, it does not allow the device to go to any sleep mode. This consumes constant power.
- **Signal quality:** Similar to cellular network, when the device is placed in a poor signal area, the location service requires more power in order to amplify the weak signal or to search for a lost signal.
- **Device position:** Materials that are around us such as walls, roofs, ... have also an impact on the strength of the received signals. In these circumstances, the receiver amplifier drains more power to increase the signal from satellites.

FM Radio

FM radio is a broadcasting method that uses Frequency Modulation (FM). It is mainly used by radio stations to share audio. Although it's not the biggest energy consumer, FM radio still drains power when it's in use. Here are some of the elements that impact this energy consumption:

- **FM radio receiver chip:** Similar to the GPS receiver, the FM radio receiver can consume much power when trying to maintain a good quality signal. Indeed, when the received signal becomes too poor, the chip drains power to properly amplify the signal.
- **Antenna:** The antenna is an essential component for all devices with a radio receiver in order to capture radio waves. As recent smartphones do not

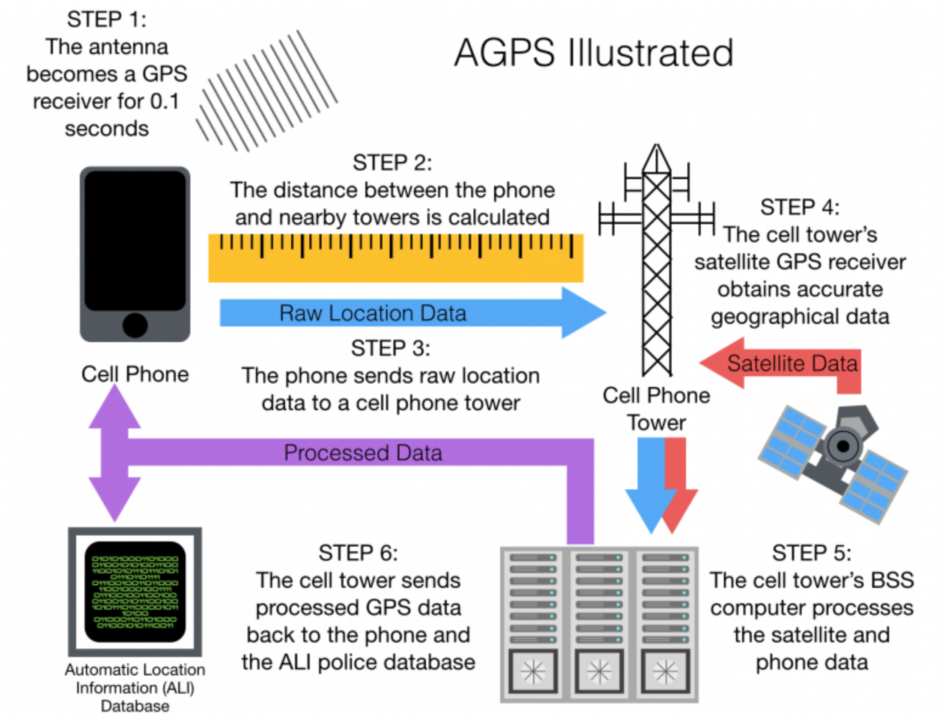

AGPS Illustrated
STEP 1: The antenna becomes a GPS receiver for 0.1 seconds
STEP 2: The distance between the phone and nearby towers is calculated
Raw Location Data
STEP 3: The phone sends raw location data to a cell phone tower
STEP 4: The cell tower's satellite GPS receiver obtains accurate geographical data
Satellite Data
Cell Phone
Cell Phone Tower
Processed Data
STEP 5: The cell tower's BSS computer processes the satellite and phone data
STEP 6: The cell tower sends processed GPS data back to the phone and the ALI police database
Automatic Location Information (ALI) Database

Figure 2.5: Assisted Global Positioning System (AGPS) principle illustrated[a], A-GPS augments basic satellite information by using cell tower data to enhance quality and precision.

[a]Illustration from www.in.c.mi.com.

have space to store an antenna, the connection of headphones is required to act as a temporary radio receiver. However, as a headphones cable is not rigid enough signals cannot always properly be received, which is why signal amplification may be needed.

Chapter 3

Smartphone Power Consumption Measurement

Now that we have analyzed the electronic components that have an impact on the power consumption of the smartphone, we review the techniques that estimate and analyze the energy consumption of smartphones using program analysis. This step is crucial to accurately identify the practices implemented in an application that are not energy-efficient. Most of the research works evaluate their findings by comparing them to the overall consumption of a smartphone using hardware measurement systems. This chapter describes techniques based on the construction of power models, tools for static analysis of programming errors related to energy consumption, solutions for estimating the cost of implementations using repositories, and tools for reporting the analysed data.

3.1 Smartphone Power Consumption Analysis and Modelling

Getting the overall energy cost of a smartphone is not the easiest analysis to carry out. There are indeed many elements involved in the chain of smartphone energy consumption. From the hardware component (Figure 3.1) through the hardware driver, the operating system layer and at last the application. The way in which the electronic component is used is not always energy-efficient [86] but it is necessary to understand where does the issue lie. This need for estimation and analysis can be supported by the generation of power models. In this approach, solutions exist for estimating and analysing the energy consumption of the elements that are part of this energy consumption chain. A fine granularity estimation requires a deep level of analysis. The solutions reviewed in this chapter focus on usage-based data, data provided by system calls and finally, data estimated at the level of a method call as well as at the level of instruction execution.

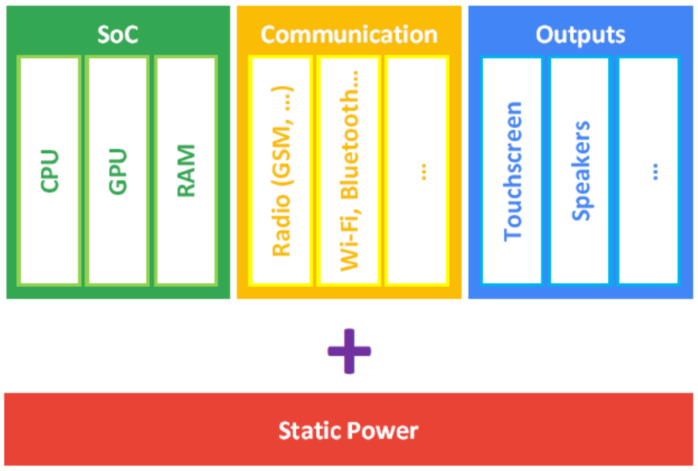


Figure 3.1: Power consumption distribution in modern mobile systems as defined by authors [35]. The System on a Chip (SoC), the communications modules and the output resources are all battery's energy consumers.

Utilization-Based Modelling

Utilization-based power models represent full system energy consumption by profiling each component's energy consumption in two steps. The first step is to collect utilization data[1] from a set of scenarios and its corresponding energy consumption. The second step is to apply linear regression analysis[2] in order to construct the power model for software energy consumption based on the collected utilization data. The trained model is then used to predict energy consumption. It can be used to locate application's hotspots[3] by reporting some of the characteristics during run-time and data collected can also prevent excessive energy consumption of the source code.

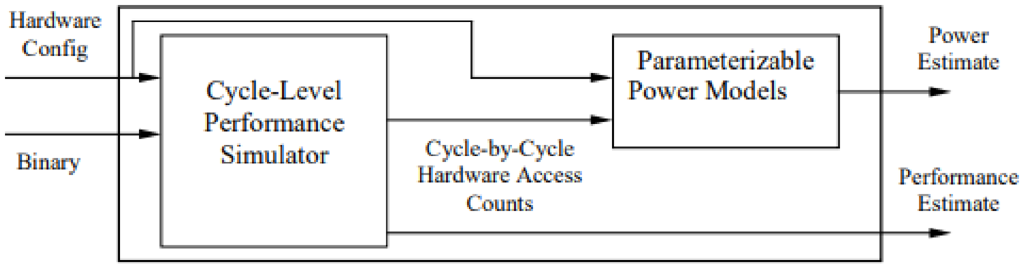
This power model is widely exploited in the domain [63, 53, 7, 36, 51, 24, 94] where *PowerScope*[53] was the first tool able to profile the energy usage by applications. It has several shortcomings, first it requires an external monitor infrastructure and second it is only dedicated for single core architecture for which only one application can be executed at the time.

Following prior work, Brooks and colleagues [28] developed a framework for architectural-level power analysis and optimizations. They are concerned about the power dissipation that issue designers of embedded or portable computer

[1]The data are collected during application run-time and allow to generate a method's Call Graph Flow (CFG) for each functionality. A call graph flow provides fine-grained details of the structure of the program as a whole, especially its subroutines.

[2]Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model. Source: www.stat.yale.edu/Courses/1997-98/101/linreg.htm

[3]Issue in software code where a wrong behaviour is discovered.



Hardware Config
Binary
Cycle-Level Performance Simulator
Cycle-by-Cycle Hardware Access Counts
Parameterizable Power Models
Power Estimate
Performance Estimate

Figure 3.2: Overall structure of the power simulator Wattch [28]. The framework uses power models to simulate performance of the CPU at cycle-accuracy level.

systems are facing. Their work, *Wattch*[28] is a power simulator that estimates CPU power consumption based on a suite of parameterizable power models. According to them, *Wattch*[28] can be integrated into a wide range of architectural simulators. As proof of concept, they have integrated their framework into the SimpleScalar architectural simulator [29] in order to provide power estimates. Figure 3.2 shows the overall structure of *Wattch*[28] and the interface between the simulator and their power models. Their work is focused on power modelling the main processor unit which they divided into four categories and thus four models:

- Array structures composed of data and instruction caches, all register files, load/store queues,...
- Content Addressable Memory (CAM) or Associative Memory which is a memory that implements the lookup-table function using dedicated comparison circuitry
- Combinational Logistic and Wires composed of functional units, dependency check logic, results buses,..
- Clocking which gathers the clock buffers, the clock wires and the capacitive loads

Even though Wattch[28] can achieve power models with high sampling rate as it is cycle-accurate (according to the results it has an accuracy of 10% and can be up to 1000X faster than prior existing layout-level power tools), it requires circuit-level power characterization and are generated by a fixed set of benchmark applications which is a major limitation and drops the accuracy of the models. Indeed, smartphone usage is diverse among different users and cannot be modeled through a fixed set of benchmarks.

On the contrary, recent related work, *Valgreen* [41] from Cupertino and colleagues, bases its models from real energy usage of individual users. It is an application's energy profiler which exploits the battery's information in order to generate an architecture-independent power model through a calibration process. It samples the power of an application in short time steps and measures the total energy spent during the execution of such application. They collect utilization statistics through the Advanced Configuration and Power Interface (ACPI)[4] at no additional cost. They also exploit operating system sensors to measure the behaviour of the application and then collect data related to the hardware usage. As machine related sensors can only measure property for the whole computing system, it cannot make distinction of running processes. However, *Valgreen* tool

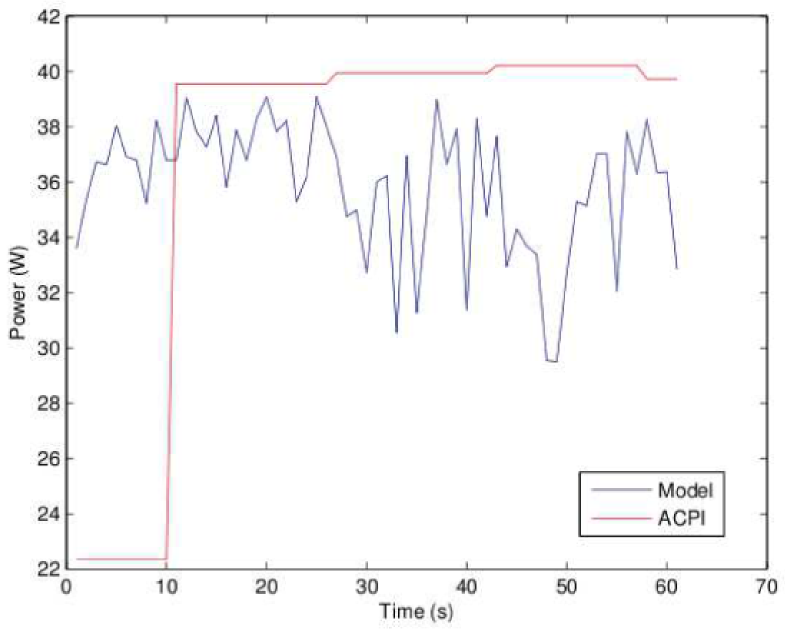


Figure 3.3: Comparison between ACPI power meter and *Valgreen*[41] linear model showing a better accuracy for estimating the real power consumed.

can link up the energy usage to a specific process using application related sensors (eg. CPU usage) and the ACPI power meter. The authors state that the CPU, memory disk and hard disk are the most consuming devices. For portables devices, the wireless network interface card and the screen can also have a great impact. Since the CPU is the most consuming device, the default mathematical model used to estimate the power consumption of applications in *Valgreen* [41] is a CPU proportional model, described as follows:

$$P_{pid} = w_0 + w_1 \sum_{cpu} \frac{t_{pid}^{cpu}}{t_{cpu} + t_{idl}} + w_2 \frac{1}{|RP|}, \forall_{cpu} \in C$$

where t_{cpu}, t_{idl} and t_{pid}^{cpu} are, respectively, the total, idle and process' CPU time, C is the set of all processors available, $|RP|$ is the cardinality of the running processes set and the weight vector $w = [w_0, w_1, w_2]$ is the set of architecture

[4]In a computer, the ACPI provides an open standard that operating systems can use to discover and configure computer hardware components, to perform power management by (for example) putting unused components to sleep, and to perform status monitoring. Source: en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface

dependent parameters [41]. Calibration process is composed of two steps. First, they retrieve ACPI sampling rate[5] in order to estimate the total duration of the data recovery. Indeed, this sampling rate is required for the second step to collect enough data for the power model input by learning all the parameters. The calibration is an important criteria for the accuracy of the model. Figure 3.3 shows the accuracy of *Valgreen* using linear model compared to the current draw data from the ACPI. Through this work, they demonstrate that a power model can have a high accuracy based on collected data from embedded power meter. Additionally, using a well-defined model can bring update frequency from less than 1 Hz (ACPI frequency) to 10 Hz. However, the accuracy stands at process level while instruction-based power model can go down at the function level and even deeper.

Another similar approach is the work of Dong and colleagues [63] called *Sesame*, which consists on a self power measurement through the smart battery interface. They offer an high rate system model without external assistance. This particularity allows them not to be hardware configuration dependent compared to other works that have dependencies of the system models and thus require a personalized model for every different mobile device. Their work achieves accuracy similar to state-of-the-art system power models using their individual sampling, PowerTutor at 1 Hz [58] , PowerBooter at 0.1 Hz [58] and Finite State Machine (FSM) at 20 Hz [9]. They demonstrated the interest of research in self energy modeling.

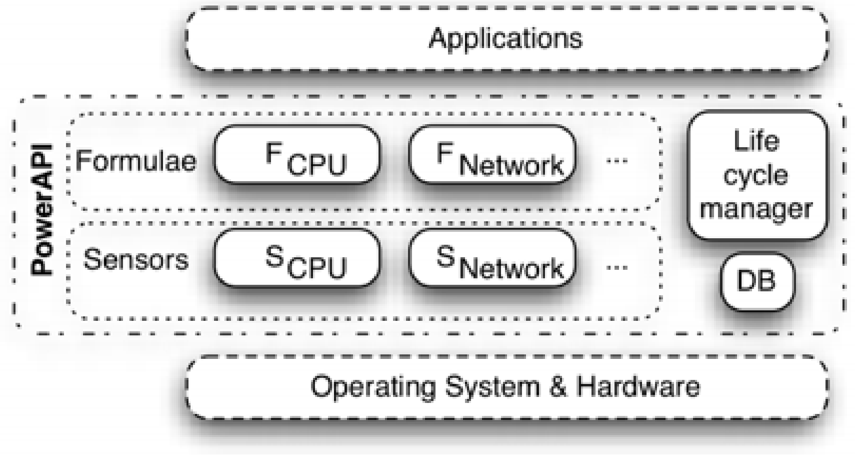


Figure 3.4: PowerAPI [7] reference architecture. The framework uses information from the operating system to estimate application's power consumption.

Related work [7] offer process-level energy information using monitoring modules built with state-of-the-art models and formulae. The power model they are

[5]A sampling rate is the frequency at which samples of an analog signal are captured in order to convert it into digital signal.

using for energy consumption computation is based on the following state-of-the-art formulae [21]:

$$E_{software} = E_{comp} + E_{com} + E_{infra}$$

where E_{comp} is the computational cost (e.g CPU processing, memory access, I/O operations). E_{com} is the exchanging date over network cost and E_{infra} is the additional cost (e.g cost of Java Virtual Machine (JVM), Operating System (OS)) [7].

PowerAPI is implemented as a system level modular library which consist of CPU and network modules and associated power models. Each system component is represented as a power module aiming to provide power information per Process IDentifier (PID). Each power module consists of two sub-modules (Figure 3.4): formula and sensor. The sensor sub-module gathers hardware and operating system related information. For example, their implementation exploits system information available in procfs [39] and sysfs[80] file systems. It means that this approach for the sub-module is however OS-dependent. The formula sub-module calculates the power consumed for each process by using information retrieved by the sensor sub-module. This sub-module is on the other hand, platform independent. However, *PowerAPI*[7] approach does not cover hardware such as memory and disk.

Although linear regression analysis is popular[7, 53, 58], linear model assume -by definition- that the variations in power consumption are linear causing an increased estimation error when it is not the case [48]. Base on this fact, some researchers [35] have recently been experimenting with algorithms such as the use of neural networks[6] to build power models on ARM architectures[7]. The technique is called a NARX non-linear[8] neural net previously proposed by Xie and al [95]. The authors show that this solution is simpler, easier to implement and above all, more suitable because changes in consumption are not always linear. They assume that the total amount of power consumption consumed by the smartphone can be split into the sum of powers consumed by each component (Figure 3.1) and a static amount of power consumed by the smartphone [48], giving the following formulae:

$$P_{Total} = P_{Static} + \sum_{i=1}^{n} P_i$$

where i is the component number (CPU, GPU, RAM...) and n is the total number of components [35]. They apply NARX neural network [95] to predict the output of time series and construct the power model using above parameters as inputs. However, this work is only applicable to a particular architecture, the ARM

[6]A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Source: /www.investopedia.com/terms/n/neuralnetwork.asp

[7]ARM is a family of reduced instruction set computing (RISC) architectures for computer processors, configured for various environments. Source: en.wikipedia.org/wiki/ARM_architecture

[8]NARX network is a dynamic neural network which appears effective in the input-output identification of both linear and nonlinear systems. Source: https://ieeexplore.ieee.org/document/7244449

processors, which limits the field of use even if the authors demonstrate that the accuracy is by 97% compared to an external measuring system. Future work will be to study the health status and the operational states of the systems.

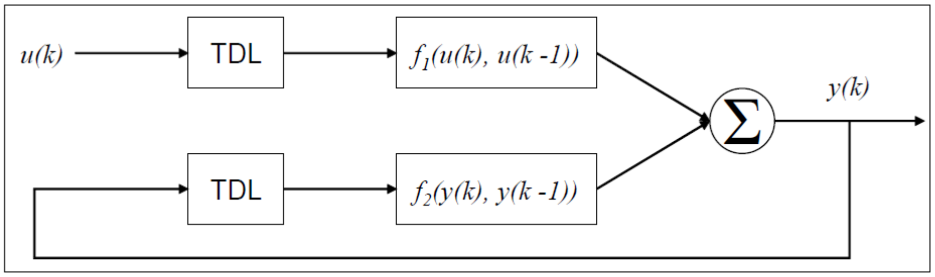


Figure 3.5: Neural network general structure for NARX nonlinear neural net used in [35], with output feedback and the Time Delay Line (TDL)

Instruction-Based Modelling

Instruction-based power models for applications in JVM are using Java bytecode[9] instructions to build energy models for software systems.

Seo and colleagues [21, 22] propose an energy consumption model for Java-based software systems running on distributed devices. This work is integrating three components: computational energy cost; communication energy cost; and infrastructure energy overhead. It shows an accuracy that falls within 5% of the actual energy cost for an application but this model is highly dependent on the hardware and JVM.

Based on this work, Shuai Hao and colleagues [86] propose a new approach for estimating the energy consumption of mobile applications. Their solution is both lightweight in terms of its developer requirements and provides fine-grained estimates of energy consumption at code level.

In this approach, the authors are using *eLens*[86], a combination of program analysis and per-instruction energy modeling that is able to estimate energy consumption to within 10% of the ground truth for a set of mobile applications from the Google Play store. In their previous work [85] they used execution traces to estimate CPU energy consumption. The application of the techniques were limited as they were only able to estimate energy for instructions. Their new paper goes further by adding more sophisticated CPU model and techniques that include other hardware components such as RAM, Wi-Fi and GPS. All this is done via Software Environment Energy Profile (SEEP). The inputs used in their approach are: the software artifact; the workload, which describes how the users will interact with the application; and the system profiles, which provides the power characteristics of the platforms by using per-instruction energy models. As shown in Figure 3.6, *eLens*[86] contains three components: the Workload

[9]Java bytecode is the instruction set of the Java virtual machine (JVM).

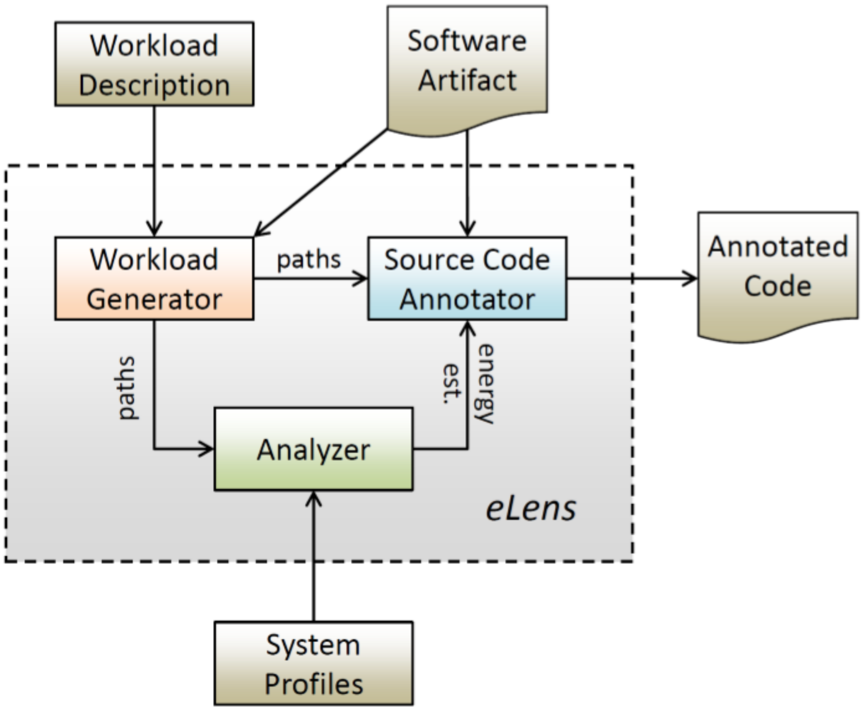
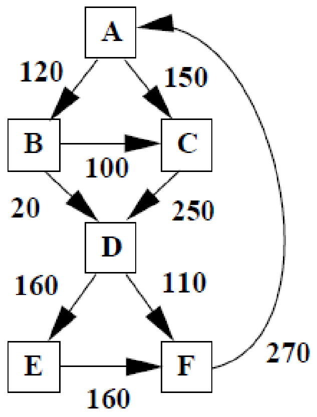
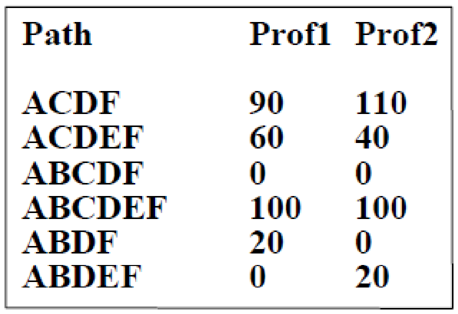


Figure 3.6: General overview of eLens's framework [86] to detect and show in source code the energy hotspots. Based on workload reports and system profiles, it is capable to find misuse of resource and annotate the source code.

Generator which has the main job of translating the workload into sets of path through the software artifact; the Analyzer is the element that will use the previously defined sets of path and the external system profile to compute an energy estimate; and the Source Code Annotator which makes a combination of both paths and energy estimate to render an annotated version of the source code. Their work integrates path profiling techniques based on the state-of-the-art T. Ball and R. Larus [90] algorithm. A path profile will determine how many times each acyclic path in a routine executes. It has a real potential for program improvement, software test coverage and profile-directed compilation. Fine-grain profiles using basic blocks and control-flow edges are broadly used for profile-driven compilation. Although their accuracy can be better, the low cost and small complexity gives the advantage to quickly profile an application. Aside of this technique, T. Ball and R. Larus [90] propose a path profile algorithm that points out accurate profiling is neither complex nor expensive. Figure 3.7 shows a commonly heuristic to select the most frequented path [81] where we can see basic blocks and control-flow edges technique giving a very disparate number of path executed times between profiling executions *Prof1* and *Prof2* resulting in bad accuracy. Ball and Larus algorithm uses a single instrumentation variable per method to record path traversed through the CFG of a method. By assigning





Path	Prof1	Prof2
ACDF	90	110
ACDEF	60	40
ABCDF	0	0
ABCDEF	100	100
ABDF	20	0
ABDEF	0	20

Figure 3.7: Illustration and description from [90], in which edge profiling does not identify the most frequently executed paths. The table contains two different path profiles. Both path profiles induce the same edge execution frequencies, shown by the edge frequencies in the control-flow graph. In path profile *Prof1*, path ABCDEF is most frequently executed, although the heuristic of following edges with the highest frequency identifies path ABCDEF as the most frequent.

weights to edges of the CFG and summing them, they are able to generate an unique path ID for each method traversed. It allows *eLens* to quickly profile application and to use the set of paths to compute an energy estimation based on per-instruction power model.

The authors used a case study to demonstrate the accuracy and usability of *eLens* for measuring power consumption of marketplace applications. Regarding the accuracy, the results show *eLens* estimation error at the whole program level is below 10% with an overall average of 8.8% and an overall average of 7.1% at the method level. Regarding the usability, the instrumentation time ranged from 4 to 14 minutes and the analysis time ranged from 6 to 17 seconds. Authors recall that in practice, the instrumentation time would be much lower because only pieces of code (e.g object class[10]) changed from an iteration to another would be instrumented as opposed to this case, where all objects class were instrumented. However, like most instruction-based power model solutions, this work is only applicable for software running in JVM. It implies limitations on the usability of such technique.

[10]In object-oriented programming, a class is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods). Source: https://en.wikipedia.org/wiki/Class_(computer_programming)

System Call-Based Modelling

System calls are the interface between applications and the OS kernel. They form an Application Programming Interface (API) for users applications to access services and devices that are managed by the OS. They are standard functions made available for user processes. Figure 3.8 presents the relationship between users applications, C library functions, systems calls and the OS kernel [84]. Researchers are investigating on how to make the relation between energy consumption and system-calls as it provides insight on how hardware and software resources are used by the application and thus could help developers to detect performance regressions dues to changes made on the source code. Contrasting with profiling energy consumption, profiling system-calls requires no external monitoring tool, it becomes easier for the programmer to track changes to system calls. Indeed, profiling system-calls is less resource intensive than setting-up specialized hardware test bed and using power monitoring tools in order to profile energy consumption.

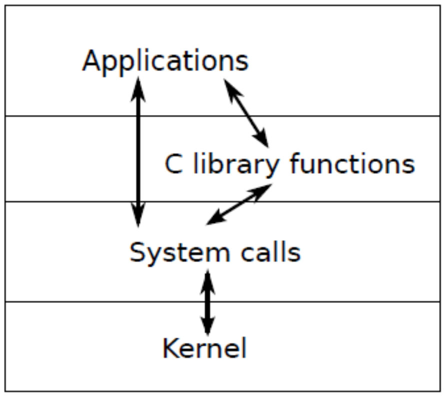


Figure 3.8: This diagram [84] shows how applications, C library functions, system calls, and the kernel interact with each other.

A system-call-based power model was first proposed by Pathal and colleagues [9] where they applied energy consumption model to estimate power consumption of applications during run-time. They showed by analysing the behaviour of components in smartphones that: several components stay in high power state for a period of time after use; system calls that are not in use can change power states and several components do not have quantitative utilization. In lights of these observations, they conclude that building energy linear model based on correlating utilization with energy consumption is not accurate. Their second approach was to build energy models by tracing system calls and generate a FSM[11] for each system call of each component. They then integrated all the FSMs to build a

[11]A finite-state machine (FSM) is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. Source: en.wikipedia.org/wiki/Finite-state_machine

FSM for each component. Finally, they gathered the FSMs to build the FSM model of the smartphone. Using this model, they can identify the current state of the system and estimate the energy consumption of an application. Their result showed an improved accuracy compared to *PowerScope* [53], which is based on linear regression modeling. They extended [8] their prior work and implemented a fine-grained energy profiler for mobiles using FSMs. Their tool can be executed on both Android and Windows Mobile operating systems.

Similar to this system-call-based power model, Aggarwal and colleagues [56] also trace system calls and correlate them with software energy consumption. However, their work uses a change model using logistic regressions to predict if a new version has significantly different energy consumption compared to the previous version. This is based on the difference in system call invocations. Their approach uses *Green Miner* [5] testbed with hardware equipment to measure the energy consumption of Android applications. *Green Miner* [5] is able to collect data across multiple application versions. This type of technique is introduced in the last chapter. The second step was to write test scripts to generate the workload as input to the application during run-time. They finally collect and analyze data. Through their work, they demonstrate that system-calls suffer from limited variability. They conclude that most system-calls are moderately related to energy consumption as the solution they found was to rely on averages to make the relationship between system calls and energy consumption. Regarding the reliability of such estimation, their evaluation on the *Calculator* and *Firefox* showed respectively an accuracy of 87,7% and 80,04%.

3.2 Detecting Energy Hotspots, Energy Bugs and Energy Leaks

An alternative or even a complementary solution to the construction of energy models is the detection of energy failures within the code of an application. These failures are mainly programming errors that are not always detected by a standard compiler. This section describes findings that would allow the programmer to be aware of the energy impacts on the way he programs, as if it was a standard programming error.

Detecting Energy Bugs

Energy bugs are defects that can have a great impact on the energy consumption of mobile applications. As some resource intensive hardware components are requested during run-time, the battery energy can be mostly consumed by the applications usage. There are two main families of energy bugs: the first one is the resource leak; it refers to a case when an application does not release the resource after use and the impact is the battery drain for no actual reason; the second one is the layout defect that refers to a poor software layout structure where for instance the layout is too deep and there are too many or ineffective widgets.

Work related to energy bugs detection focus on detecting either background programs [92, 91] or foreground ones such as user interfaces [45].

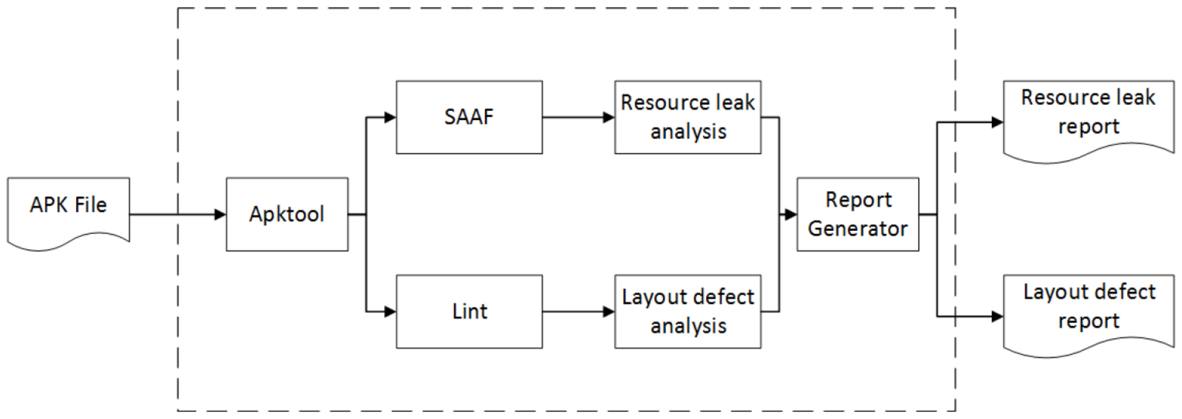


Figure 3.9: Illustration of the framework of Energy Bug Detection [44]. Two software analyzing tools are used to detect resource leaks and layout defects.

Research [44] focus on detecting resources leaks and layout defects. Their approach, SAAD[44] which stands for Static Application Analysis Detector (SADD), is able to detect and to report applications resource leaks using inter-procedural and intra-procedural analyses. In this approach, they integrate several energy bugs analysis and detection tools such as Apktool[12], SAAF[47] and Lint[60]. As shown in Figure 3.9, the input of their framework is an APK[12] file which is decompiled into Dalvik bytecode[13] in order to generate the manifest configuration file of the application. This file includes declared components required to establish relationship between calls to abstract them into a component call graph. De-compiled application is then used as input for the open sourced static Android analysis framework SAAF[47] which encapsulates it into its data models (Instruction model, BasicBlock model, Method model and SmaliClass model). In parallel, *SAAD*[44] uses the static analysis tool Lint[60] to detecting performance problems of the structure code. It takes as inputs the Android project source files and an XML[14] file in which severity levels of problems are defined. A detailed report is generated based on Lint[60] analysis. Jiang and colleagues contribution resides mainly in what comes after; they implemented two modules: the layout

[12]An APK file is an Android Package file that's used to distribute applications on Google's Android operating system.

[13]Dalvik bytecode is similar to Java bytecode. The two most notable differences are: Dalvik is register based rather than stack based and the local registers are untyped. These differences are reflected in their two instruction sets. Source: www.sciencedirect.com/science/article/pii/S0167642313003304

[14]Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. Source: en.wikipedia.org/wiki/XML

defect module which basically integrates the output from Lint but also filter the report in order to extract energy defects issues. This filter sub-module has a set of filter rules that points out which issue is an energy problem and thus which one is relevant; the resource leak detection module integrates the component call graph to explore the function call path with resource applying and releasing and further extract a set of executions paths. Each path flow is checked to see whether or not the acquired resource is released after use. Post-analyis of both modules output report result where energy defects issues and resource leaks are summarized. Although their evaluation shows good results, it is only based on common Android system functions analysis and thus does not cover all the component call of an application.

In conjunction with CPU energy management, ineffective use of sensors and their data can also cause severe energy leakage. Chang Xu and al [62] focus their analysis on sensor's utilization in Android applications. Their approach automatically detects sensor's data at different states and reports information to help developers locate energy leaks. The analysis tool called GreenDroid simulates the runtime behavior of applications by taking as input the application's Java bytecodes (*.class format) and configuration files (*.xml format) and construct application execution model. Under this approach, they implement their solution on top of Java PathFinder (JFP)[15]. GreenDroid leverages JFP model checking functionality to generates user interaction sequences to create the execution paths. Moreover, they use dynamic tainting-based techniques[16] for analyzing sensors data. The Android application's data objects are tainted with a unique mark before being fed to to GreenDroid. They then track the sensors usage data by variables in each bytecode instruction. Finally, the constructed application's execution model reports sensors data power consumption at specific point in the source code. The framework show good results but requires JVM and analysis can only be executed Android applications. Furthermore, their solution does not detect misuse of wake locks[17] on sensors.

3.3 Tools and Applications for Power Consumption Monitoring and Analysis

This chapter discusses the main software tools and methods for analyzing, estimating, and reporting the energy consumption of smartphones.

[15]Java Pathfinder (JPF) is a system to verify executable Java bytecode programs. JPF was developed at the NASA Ames Research Center and open sourced in 2005. Source: https://en.wikipedia.org/wiki/Java$_{pathfinder}$

[16]Software taint analysis detects overwrite attacks which include most types of security breeches.

[17]Wake locks are power-managing software mechanisms used to indicate the device to stay on.

Mining Software Repositories

Mining Software Repository (MSR) research aims to increase awareness amongst developers in order to base their decisions on data mined from software repositories such as version control systems, bug trackers and project documentation. Many researches on software power consumption are made but only a few focuses on power consumption related to software change. Moura and colleagues [68] conducted a study on 2,000 commits on several open-source projects in Github[18] where they found out that 371 of them were probably dedicated to save software energy consumption using energy-efficiency solutions such as frequency scaling[19] and levels of idleness[20]. Yet this conclusion is based on programmers' commits that were manually analysed.

One relevant work is Abraham Hindle paper [3] which presents an abstract methodology for measuring and correlating power consumption of software across multiple commit versions. *GreenMining*[3] measures and extracts power consumption information relevant to software change. Their methodology is as follows (extract from paper [3], pp. 3 - 5):

- Choose a software product to test and which context it should be tested in.
- Decide on the level of instrumentation and on the different kinds of data recorded, including power measurements.
- Choose a set of versions, snapshots or revisions, of a software product to test.
- Develop a test case for the software that can be run on the selected snapshots and revisions of the software.
- Configure the testbed system to reduce background noise from other processes.
- For every chosen version:
  - Run the test within the testbed and record the instrumented data.
  - Compile and store the recorded data.
  - Clean up the test and the testbed.
- Compile and Analyze the results.

They used external Alternative Current (AC) power monitor "Watts Up? Pro" hardware device to measure power consumption of the system which is able to sample at a frequency of 1 Hz. System activity information such as CPU, disk and memory were also required in order to model the energy cost of targeted application. In this approach, they used System Activity Report (SAR)[43] which is a system activity report tool provided on Linux [61] platforms. Their evaluation was to first compare the power consumption of different Firefox commit branches and second to analyse Vuze[21] power consumption revision-by-revision. This work

[18]www.github.com

[19]See Section 1.4

[20]Levels of idleness refers to the management of idle states of electronic components

[21]Vuze is a BitTorrent client coded in Java

demonstrates the feasibility of using green mining methodology to point out power relevant behavior across multiple versions of an application. Indeed, they show that software change induce changes in power consumption but on the other hand, they also found out that results depend greatly on the structure of the tests executed and thus changes observed are not highly accurate.

Their prior work [4] was a prospect of extended possibilities mining repositories can offer, it did not analyze source code or relate software changes to power consumption. Related work [2] also explores mining techniques yet to estimate energy consumption of application using traces on Windows Phone 7 operating system.

Energy Efficiency Repositories

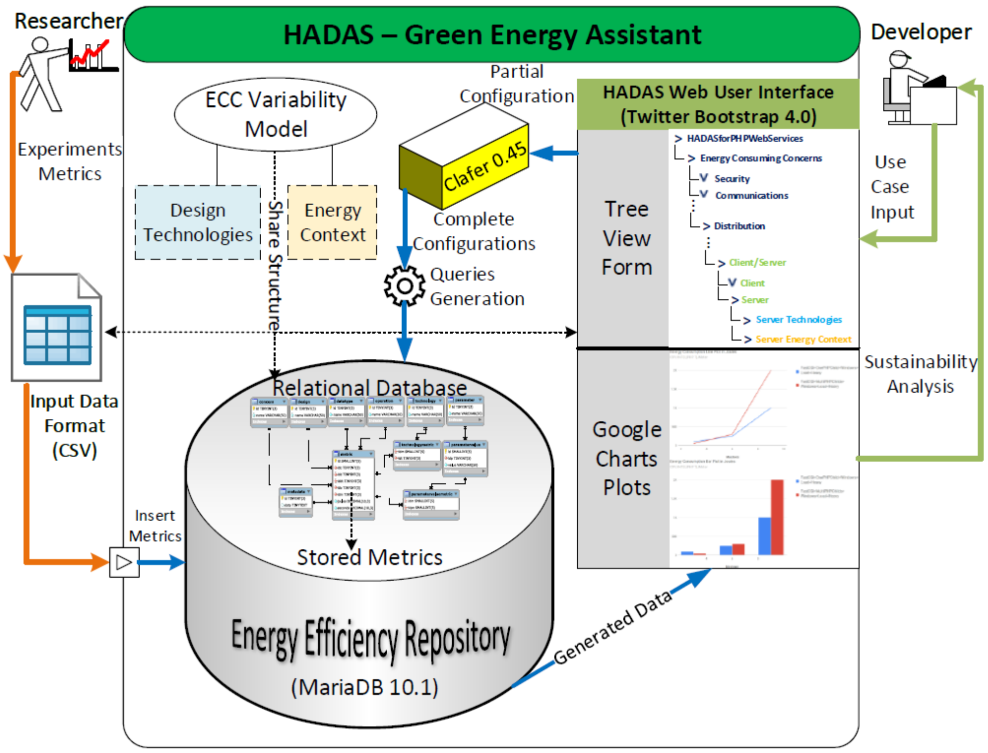


Figure 3.10: Overview of HADAS toolkit [73] where two roles for using the solution are clearly defined: developer and researcher. Researchers provide power energy input data while developers rely on power consumption data to select the best energy-efficiency implementation according to their needs.

The tools mentioned can therefore help to ensure that an eco-friendly application is designed through the analysis of commits, but this information on the cases detected and their findings remains stored in the programmer's head or, in the best case, in the coding style of a software development company. They are not shared to the developer community so they cannot become standardized solutions. D-J Munoz author [73] is aware of this lack of sharing. He presents a solution using a Software Product Line (SPL)[22] approach called HADAS (Figure 3.10). The purpose of this tool is to connect the software community that includes researchers and developers. This solution relies on a MariaDB[23] relational database which stores and categorizes the energy-efficiency implementation designs. The information is categorized into structure layers (Energy consuming concerns, Design variants, Implementation alternatives, Energy context, ...). The researcher produces data for the energy efficiency repository in the form of a design solution with an attractive energy performance. To achieve this, he fills in a template containing many fields that define the technologies used, the energy cost, etc. of the proposed design. The developer has at his disposal a web interface where the data is displayed in tree view. He can select implementation alternatives according to his needs. All data relating to the relevance of the design, i.e. its energy consumption, are displayed. Unfortunately their system is not fully automated and seems to be highly related to web development based design. Authors continue working on the toolkit with new features, improvements and a more user-friendly interface [69, 70, 71, 72].

[22]A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. Source: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513819

[23]mariadb.org

Chapter 4

Smartphone Power Consumption Optimization

Analyzing and estimating the energy consumption of applications is a big step but in order to make the applications energy-efficient, it is necessary to optimize the resource usage. Many of the techniques proposed for the analysis provide a lead so that the programmer can reduce the energy cost. In these situations, optimization depends on how the data will be interpreted and used during software development. There is however a whole range of improvements that are possible either on hardware low-level power management or on applications where automatic optimization techniques can be applied. This section describes the optimization of operations allowed on hardware components such as display, computing units, task offloading techniques and code refactoring techniques.

4.1 Device Operation Optimization

Newer smart-phones include several components that enable to perform mathematical operations. They are becoming more and more powerful and some of them can be dedicated to certain types of operations while others are able to perform tasks in parallel. However, the maximum performance provided by these components is not always required [23] and could be tailored to reduce power consumption. The proposed research covers these topics that are considered important and for with a possible future in the field of energy-efficiency.

Dynamic Resource Allotment

Parallel running processes are a common feature in recent OS, with some running in the foreground and others in the background not necessarily requiring the same resources. Mukherjee and Chantem's [6] offer a solution to minimise the energy cost of these processes that run in the background. Their work relies on a framework that is able to manage the resources at system-level. The targeted components are the processor and the memory, more precisely the voltage and frequency of the computing unit as well as the memory bandwidth[1]. The idea is to adjust these parameters in real-time on the basis of a characterization that takes the form of a series of different configurations which are selected according to the current needs. The following solution is composed of two-step procedure (Figure 4.1).

They run applications side-by-side for the profiling phase which is executed offline. The profiler calculates the average performance data and energy consumption data for an application for each CPU frequency and memory bandwidth. This is stored as a tuple (CPU frequency, Memory bandwidth) and represents a system configuration (Figure 4.2). Such information will assist in decision making when managing the energy consumption through the online controller. The second step consists of an online framework which is periodically executed

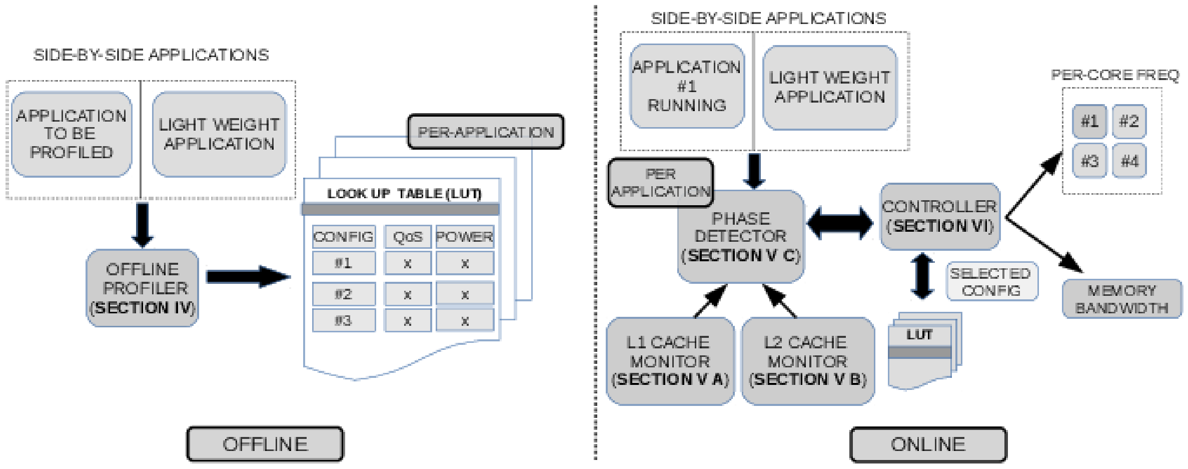
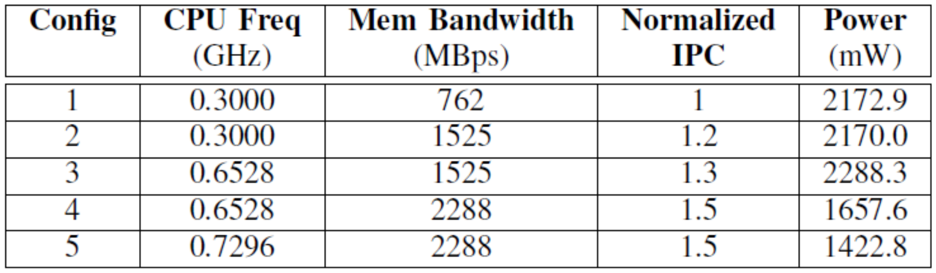


Figure 4.1: The proposed framework [6] is decoupled into running applications side-by-side in offline profiling phase (left), and an online framework to select the best energy-performance configuration (right).

for selecting the best energy-performance configuration (energy-efficiency with less performance degradation). This module relies on phase detection combined with the profiling methods. A phase transition is identified when a computation intensive phase is followed by a memory intensive phase. CPU intensive phase are monitored with existing CPU activity tool Perf [33] while memory trafic monitoring is realized with an intra-monitoring mechanism developed for the study. The memory intensive phases are monitored from CPU memory caches (L1 and L2). The information is then used to define usage patterns of the application and select the best voltage and frequency level of associated core settings for the processor; and memory bandwidth to get the most energy-efficient configuration without sacrificing the performance target.

The proposed framework is also able to progressively reduce power consumption

[1]Memory bandwidth is the rate at which data can be read from or stored into a semiconductor memory by a processor. Source: en.wikipedia.org/wiki/Memory_bandwidth



Config	CPU Freq (GHz)	Mem Bandwidth (MBps)	Normalized IPC	Power (mW)
1	0.3000	762	1	2172.9
2	0.3000	1525	1.2	2170.0
3	0.6528	1525	1.3	2288.3
4	0.6528	2288	1.5	1657.6
5	0.7296	2288	1.5	1422.8

Figure 4.2: Example of application's Look Up Table (LUT) generated for system configuration [6] with the relation bewteen CPU Freq, Memory bandwidth, IPC[a] and power consumption expressed in mW. In this example, the energy-efficient configuration for the application's resource needs is the Config. 5. Lowering the CPU frequency can cause more energy consumption as it can require more time to perform the application's operations.

[a]In computer architecture, the term Instruction Per Cycle (IPC) refers to one aspect of a processor's performance: the average number of instructions executed for each clock cycle. It is the multiplicative inverse of cycles per instruction. Source: en.wikipedia.org/wiki/Instructions_per_cycle

of newly installed applications even if no resource usage data is yet available. To achieve this, the offline profiles are browsed in order to find the closest match and select the most similar profile.

As the use of this technique does not require to modify code application, it is applicable to both open-source and proprietary applications. Authors claims it can achieve up to 31 % energy reduction compared to existing techniques and default Android Governor [11]. However, their work does not yet make it possible to manage several applications running on the same processor. They are only able to characterize the workload, adjust the voltage settings and memory bandwidth for one application at a time.

S. Li and S. Mishra's [59] work also focus on multi-cores optimization by providing a middle-ware layer capable of dynamically schedule an optimal number of cores. It uses same dynamic profiling and CPU frequency adjustment of each of the core based on CPU load and performance. Moreover, the tool benefits from user's usage and data for battery's charge and discharge time. For this study, they explore scheduling algorithms in order to trade-off energy consumption, performance, and user experience. Three algorithms are proposed for which they believe it can achieve good balance between the above criteria. Unfortunately, their paper is not publicly available which limits any exhaustive evaluation.

Task Parallelism and Scheduling

Some other research are also being explored in order to reduce both CPU and GPU power consumption using Dynamic and Voltage Frequency Scaling (DVFS) policy.

This solution is commonly employed to reduce energy and power consumption for applications by tailoring its needs. It works on two power saving techniques: the dynamic frequency scaling; and the dynamic voltage scaling[2]. Research work [87] claims that most of the time, some extra frequency level is used in applications, even if DVFS techniques are already applied. Recent work on dynamic parallelism requires costly dedicated hardware and relies on greedy algorithm for decision making. As the actual application needs could be dynamically analysed, they propose to select optimal parallelism, voltage and frequency to guarantee high power efficiency. To achieve that purpose, they use recent work on dynamic parallelism with the conventional DVFS and make the parallel choices autonomous from hardware.

In memory's power management field, [82] used the Phase Change Memory (PCM)[3] but combined it to the RAM optimizations which was already applied to personal computer systems. Indeed, they applied the same power memory management to smartphone hardware that was originally proposed for desktops and servers. Their result show that the state-of-the-art Power-Aware Virtual Memory (PAVM) mechanism has interesting power efficiency. The result is high-efficient compared to the standard system with no energy management. Moreover, they observe no performance degradation on their hybrid mechanism while they degradation on both and PCM technologies are identified. Their analysis suggests that hybrid memory may be a better choice for future smartphone design.

Parallel management of communication data is another field that some researchers are studying [27]. They believe that parallel communications require less energy than their stand-alone execution. Their work analyzes the possibilities for scheduling communications in order to reduce energy consumption. In this approach, they identify two types of communications to manage a service, the Delay-Tolerant Services (DTS) and the Real-Time Services (RTS). The first type groups services that do not require instant updates when they are not in the foreground. The second type includes services that have a real-time need such as a video streaming or a phone call. Based on this distinction, they propose to schedule the first group opportunistically in parallel with RTS. They exploit the freedom in the scheduling policy for DTS services in order to execute them at minimum cost. Evaluation of pairing strategy benefits is performed on simulated traces for 3G, 4G and Wi-Fi networks. Results show 10 % to 15 % of energy saving for file uploading/downloading task in parallel with other real-time services, compared to sequential operations. However, no evaluation was performed on real communication data and there is currently no concrete application found.

[2]See Section 1.4

[3]Phase-Change Memory is a type of non-volatile RAM that stores data by altering the state of the material. Different from RAM, data can be overwritten without having to erase it first.

4.2 Optimizing Device Display

Display systems answer to a significant portion of the power consumed by the smartphone despite the emerging technologies for low-power display. Indeed, constructor innovations are moving in this direction, focusing on reducing the size of components and their energy consumption while increasing their performance (contrast, brightness, life-time, ...). This chapter describes techniques for optimizing the display system to make it even more energy-efficient. Research in this area [42, 25, 14], focuses on better management of the power supplied to this component, and on solutions to reduce the brightness of the display performing image processing. To this end, the main display technologies used for these purposes are the LCD and Organic Light-Emitting Diode (OLED).

Dynamic voltage scaling and dimming of OLED panel

Some screen technology such as LCD require more power than current OLED[4] screen where a backlight system is necessary. Indeed, LCD screens use high intensive backlight to illuminate their pixels whereas OLED screens pixels are self-illuminating using organic light emission material (Figure 4.3). In this approach, researchers try to reduce power consumed by the display, especially the backlight as it is the major factor responsible for this power drain. OLED displays consume low power for high quality images but it is still a dominant power consumer.

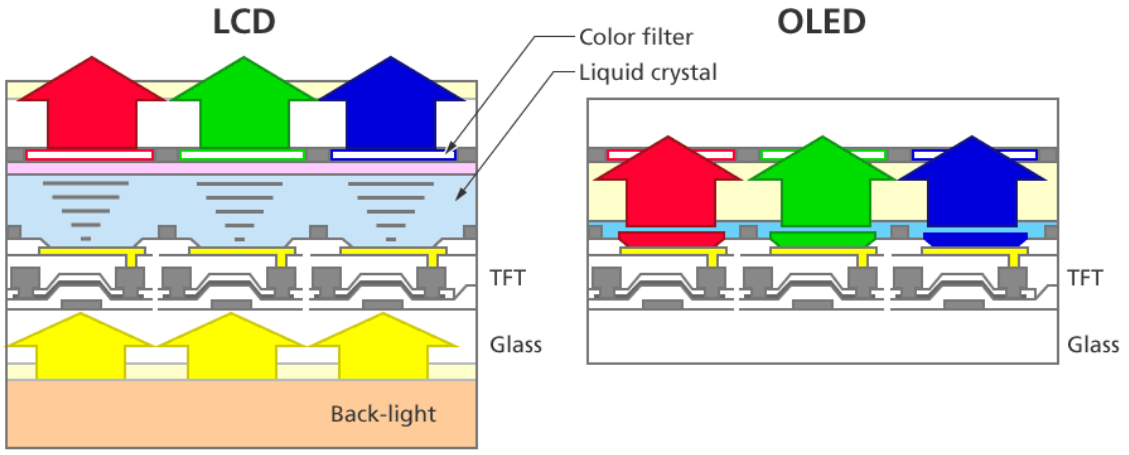


Figure 4.3: LCD layer technology (left) and OLED layer technology (right) [76]. A backligth system is required on LCD screens while OLED screens uses organic light emission material.

Interesting work uses tone mapping techniques[5] ([14] on mobile games), ([25]

[4]An organic light-emitting diode (OLED or Organic LED), is a light-emitting diode (LED) in which the emissive electroluminescent layer is a film of organic compound that emits light in response to an electric current. Source: en.wikipedia.org/wiki/OLED

[5]Tone mapping is a technique used in image processing and computer graphics to map one set

on video streams) in order to dynamically increase the image brightness on LCD displays. This allows a LCD back-light level reduction while preserving the image contrast. Anand and al [14] demonstrate it in their work on reducing energy consumption of a smartphone while running mobile games. The solution is based on gamma correction[6] applied to tone mapping. The gamma correction is an image process widely available in all graphic implementation (OpenGL, X11, game rendering engines, etc) but also as an hardware-assisted function found in many modern GPUs [77]. The system they implemented requires inputs to calculate and apply the tone mapping technique.

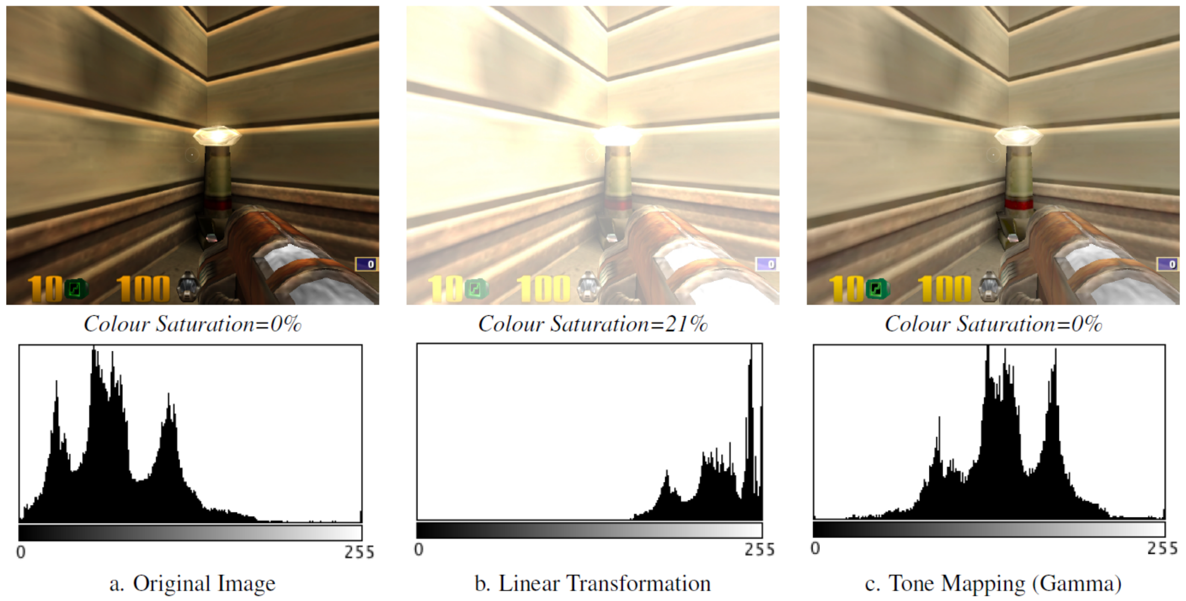


Figure 4.4: Results showing close histogram to the original image (a) using gamma correction (c) in order to reduce back-light level [14].

They first estimate the image brightness through sampling methods where they select about 2000 samples pixels (1 out of every 20 pixels from the image). For each pixel, they compute its brightness with a function that gets Red (R), Green (G) and Blue (B) colour values. They then compute a weighted average of all the pixel brightness so that the image brightness value is recovered. This final value is categorized on a fourteen level scale. The next step was to define the gamma threshold values for the fourteen brightness levels. The gamma thresholds calibration was conducted through a user study involving 10 participants. Each potential user selected the lower limit and the upper limit of the gamma correction to be applied for which there was little or no loss of overall image contrast. In

of colors to another to approximate the appearance of high-dynamic-range images in a medium that has a more limited dynamic range. Source: en.wikipedia.org/wiki/Tone_mapping

[6]Gamma correction, or often simply gamma, is a nonlinear operation used to encode and decode luminance or tristimulus values in video or still image systems. Source: en.wikipedia.org/wiki/Gamma_correction

other words, the range was not to contain any cases where the image quality was impacted, thus the quality of the game-play retained. The defined values are then computed into their run-time algorithm (Figure 4.5), that basically calculate the gamma and backlight level settings to apply for a given image brightness level.

Their analysis on two different games shows that it can save up to 68 % of the display power without significantly affecting the perceived image quality. However, this technique is not supported on Android smartphones for some less recent versions (e.g, with no powerful GPU such as the Nvidia Tegra [77]). In fact, those smartphones do not have a native software interface for the use of gamma correction. Regarding the hardware, it does not support direct access to these image processing operations. To overcome this limitation, they have managed to apply similar but limited techniques using alpha compositing methods[7], made available through OpenGL[8] graphical libraries. Furthermore, the tests conducted on old LCD screens were successful, while those ran on Active-Matrix Organic Light-Emitting Diode (AMOLED) displays showed less energy-efficiency results. They finally hope that the gamma correction methods will be supported on most new smartphones.

A more recent work proposes a video classifier based on the same dynamic tone mapping for OLED screens [25]. They found out that there is a similarity in power consumption among video streams of the same category. They use Hidden Markov Model (HMM) [46] to classify videos files into groups with a similar energy footprint. However, their findings are currently not publicly available and is limited to a Proof Of Concept (POC).

Other studies demonstrate that organic LED displays are becoming more and more popular among smartphones and have good potential for improving power consumption. Researchers such as Kim and al. [31] are trying to apply techniques to dynamically adjust the voltage supplied to these displays. Their solution is based on Dynamic Voltage Scaling (DVS) [37] techniques of the OLED panel, similar to studies that focused on improving process units energy cost [87, 59, 6]. Their work use characteristics of OLED drivers, especially the driver transistor and the internal resistance where they apply respectively an Amplitude Modulator (AM)[9] and a Pulse Width Modulator (PWM)[10]. The current applied on OLED cells determines its luminance. By adjusting the cell voltage, they can adjust the

[7]In computer graphics, alpha compositing is the process of combining one image with a background to create the appearance of partial or full transparency. It is often useful to render picture elements (pixels) in separate passes or layers and then combine the resulting 2D images into a single, final image called the composite. Source: en.wikipedia.org/wiki/Alpha_compositing

[8]www.opengl.org

[9]AM is a modulation technique used in electronic communication, most commonly for transmitting information via a radio carrier wave. In amplitude modulation, the amplitude (signal strength) of the carrier wave is varied in proportion to that of the message signal being transmitted. Source: en.wikipedia.org/wiki/Amplitude_modulation

[10]PWM is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. Source: en.wikipedia.org/wiki/Pulse-width_modulation

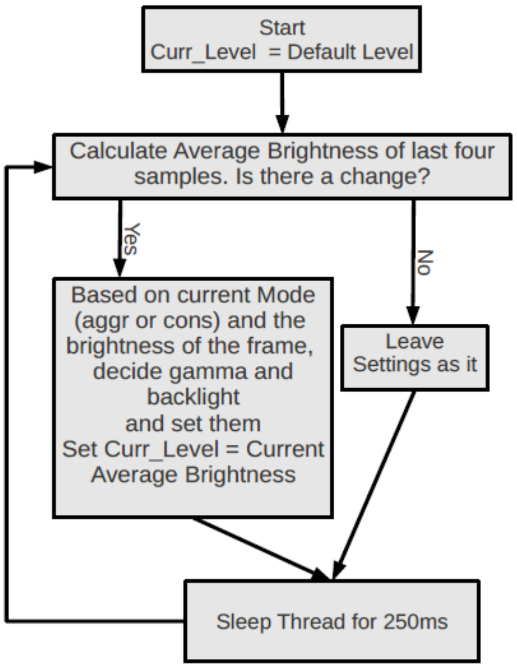

Start
Curr_Level = Default Level
Calculate Average Brightness of last four samples. Is there a change?
Yes
No
Based on current Mode (aggr or cons) and the brightness of the frame, decide gamma and backlight and set them Set Curr_Level = Current Average Brightness
Leave Settings as it
Sleep Thread for 250ms

Figure 4.5: Flowchart of gamma correction decision engine [14]. Based on current image's average brightness, the algorithm decides if it is worth it or not to apply new gamma correction and change backlight level.

cell current as,

$$P_{cell} = I_{cell} V_{cell}$$

where P_{cell} is the power given to the cell, I_{cell} is the current to the cell and V_{cell} is the voltage to the cell.

The AM and PWM drivers are used to alter the power supplied to the OLED panel which enables the reduction of its energy consumption. As the power supplied is lowered, its luminance is also reduced. However, this solution can cause image distortion. To remedy this, they apply image processing based on the human-perceived color space. Their work is different from the previous ones because even if the color of the pixels has an impact on energy consumption, changing this color degrades the quality of the image [31]. Their prototype showed 52.5 % energy saving on best case but sometimes, the luminance distortion is unavoidable therefore they sacrificed the display quality to save a certain amount of power consumption. Moreover, their solution relies on OLED hardware only, specifically on AM and PWM drivers, which limits the field of application.

4.3 Optimizing Application Design for Energy Efficiency

Despite the many proposed improvements in smartphone resource management, the way in which applications are designed remains a key element in ensuring appropriate power consumption and energy-efficiency. Tools are proposed to analyze and therefore better understand where the energy hot-spots are in a software application (see Smartphone Power Consumption Measurement chapter) but these require an effort from the beginning and throughout the design to manage these issues. There are, however, other solutions that aim to improve the energy efficiency of an application that has already been designed. This chapter reviews and describes current trends in this domain.

Task Offloading

Mobile Cloud Computing (MCC) provides the execution of mobile applications on external resource to the mobile device. It manages to offload tasks to a remote server, saving energy consumption of the smartphones. Connectivities of mobile devices offer immediate access to available computing, storage and communications on commercial clouds. However, mobile applications that benefit from the cloud computing cannot be developed as a monolithic process, it has to be split in traditional client-server paradigm.

We focus our literature review on solutions offered to optimize application not designed to take benefit from the cloud computing. To address this problem, there are research done in automatic partitioning[11], migration and execution. One related recent work is MAUI[38], which partitions applications using a framework that combines static program analysis with dynamic program profiling and optimize execution time or energy consumption using a optimization solver. MAUI[38]'s framework decides at run-time which methods should be remotely executed through energy-aware decision engine. However, this approach has some disadvantages, for example it requires the help of the programmer in order to annotate methods as *remotable*, consequently, the application needs to be written in a non-standard way.

Similar to *MAUI*[38], Chun and colleagues [15] present an approach able to partition portions of the execution but also provides automatic transformation of mobile applications to benefit from the cloud. Authors think that it can be worth to pay the cost for sending the relevant data and code from the device to the cloud. Portions of the application execution is moved onto device clones operating in a computational cloud (Figure 4.6). *CloneCloud*[15] design goal is to take programmer out of business of application partitioning by rewriting automatically an unmodified application executable. Their partitioning mechanism aims to pick which part of the application will continue to be executed on the mobile device

[11]It refers to the process for automatic detection of pieces of code eligible for migration and execution on a remote server.

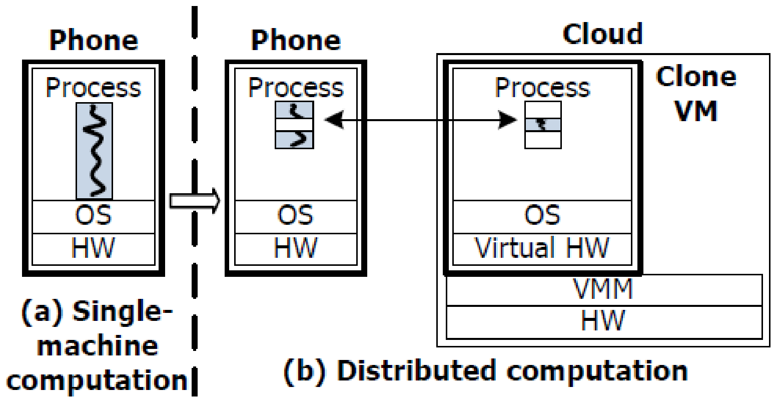


Figure 4.6: CloneCloud system model [15]. CloneCloud transforms a single-machine execution (mobile device computation) into a distributed execution (mobile device and cloud computation) automatically.

and which is migrated to the cloud. They submitted their algorithm to three main constraints in order to generate a legal partition: "Methods that access specific features of a machine must be pinned to the machine; Methods that share native state must be collocated at the same machine; Prevent nested migration" [15]. The output of this mechanism is a choice of execution points where the application migrates part of its execution and state between the device and the clone. They also integrate migration mechanism for partitions previously defined to be executed within a Virtual Machine (VM). This process operates at the granularity of a thread allowing a multi-thread application to offload functionalities, one thread-at-a-time. Basically, when a thread reaches an execution point, it will be suspended with all its states packaged and sent to the virtual machine where the process will be executed until reaching the end of the execution portion. The thread will be then packaged with the new states and sent back to the mobile application into the device. Application thread is suspended during migration process and does not block other threads unless they require to access specific resource held by the migrated thread. Their evaluation shows that their able to achieve basic augmented execution of mobile applications on the cloud. They say their solution can speed up to 20x execution time and reduce energy up to 20-fold. Yet there are limitations dues to his inability to migrate native state and to export unique native resources remotely.

Additionally to previous works, a recent solution [75] brings an interesting element which is to consider the variable capacity of the wireless network over time (Figure 4.7). Their paper presents a framework called User Level Online Offloading Framework for Mobile Edge Computing (ULOOF) which is a lightweight system that takes benefit from edge computing facilities. This application tool is a computation offloading framework that transfer and execute methods calls from a user application to a computing cloud server. It also equipped with a decision

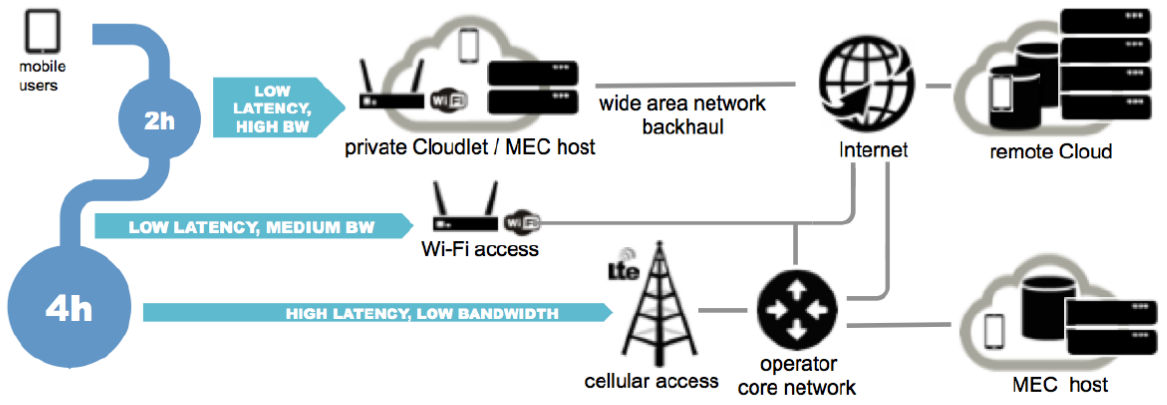
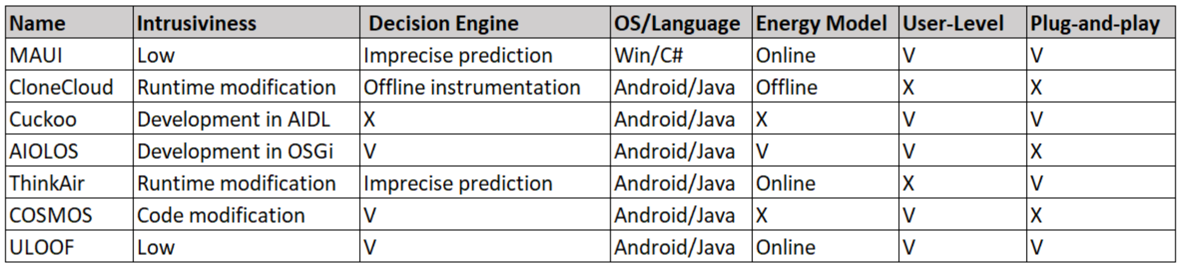


Figure 4.7: Nogueira and al [75] defined mobile computation offline scenarios to consider the variable capacity of the wireless network over time.

engine that minimizes remote execution overhead and requires no modification on the device's operating system. This tool is divided into two components, the instrumentation component; and the remote execution platform. Nogueira and al described it as follows (extract from paper [75], p. 4) :

- In the mobile device, the *instrumentation component* instruments the candidate methods for offloading. Whenever such methods are called, it intercepts the call and makes execution time and energy consumption estimations for both local and remote method execution cases. Then, the decision engine chooses whether to execute the method locally or remotely based on the estimation.
- The *remote execution platform* takes care of the remote execution of the offloadable computation, by means of a connector module. It executes the requested offloading and returns the result to the mobile device.



Name	Intrusiviness	Decision Engine	OS/Language	Energy Model	User-Level	Plug-and-play
MAUI	Low	Imprecise prediction	Win/C#	Online	V	V
CloneCloud	Runtime modification	Offline instrumentation	Android/Java	Offline	X	X
Cuckoo	Development in AIDL	X	Android/Java	X	V	V
AIOLOS	Development in OSGi	V	Android/Java	V	V	X
ThinkAir	Runtime modification	Imprecise prediction	Android/Java	Online	X	V
COSMOS	Code modification	V	Android/Java	X	V	X
ULOOF	Low	V	Android/Java	Online	V	V

Figure 4.8: Comparison between state of the art frameworks for task offloading. ULOOF [75] resolve some of the main constraints of prior techniques.

The network bandwidth estimation, which is a significant added value compared to previous techniques, is performed by the decision engine who compute this information into its calculations to decide whether or not to transfer a method on the basis of the execution time required and the energy cost of the method. Their technique seems to have successfully managed the prior solution constraints (Figure 4.8) and can even lower execution time by 50 % while offloading up to 73 % of the computations of an application. Although the framework seems efficient and meets many needs while covering the major issues of prior solutions, it requires to de-compile and re-compile the target application to mark the candidate methods to the offload. This last step creates a modified APK file to integrate the offloading logic into the application. This last point brings a constraint of use but opens opportunities to bring an energy optimization solution to applications whose development has ended up.

Code refactoring

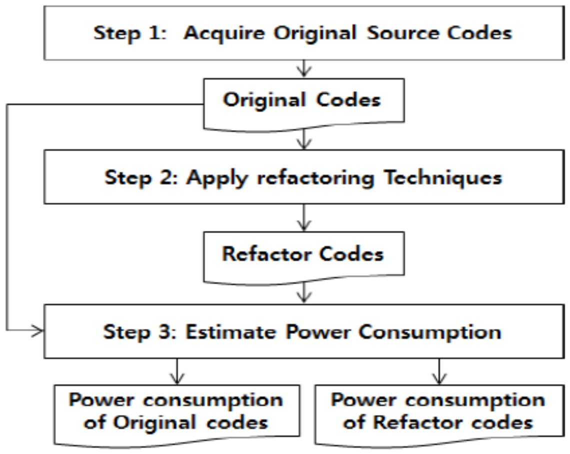
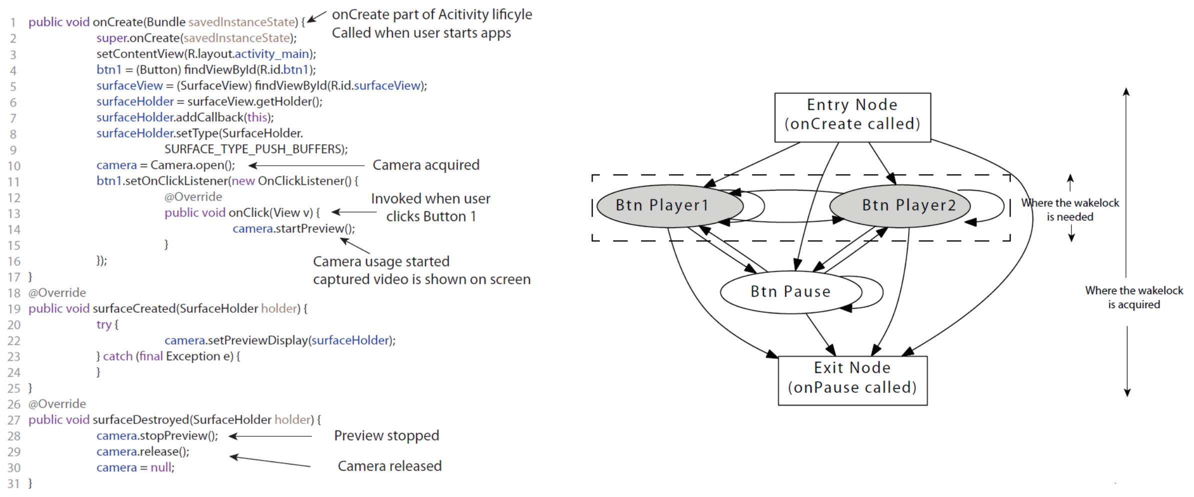

Step 1: Acquire Original Source Codes
Original Codes
Step 2: Apply refactoring Techniques
Refactor Codes
Step 3: Estimate Power Consumption
Power consumption of Original codes
Power consumption of Refactor codes

Figure 4.9: Sequence that represents the steps for the refactoring process and power consumption comparison between original and refactored code [55].

Code refactoring techniques are mainly used to improve maintainability of software to extend its lifetime. Their method focus on restructuring existing computing code without alternating its external behavior. However, some research[55] observed that power consumption is not considered in the refactoring process although some work [96, 54] used code refactoring techniques to enhance software quality attributes like performance or reliability. The energy carefree process can result in a refactored code consuming more power than the original code. Some suggest [66] that energy-efficiency shall be a dominant factor in software quality process.

In this approach, [55] compares code refactoring techniques in order to demonstrate if they can support energy-efficient software generation or not. Their work analyses M. Fowler's 63 [66] refactoring techniques as they provide general common concepts and also cover a wide range of industrial practices to software refactoring. Figure 4.9 illustrates their estimation process of power consumption for the 63 refactoring techniques. To estimate the power consumption of different



```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn1 = (Button) findViewById(R.id.btn1);
        surfaceView = (SurfaceView) findViewById(R.id.surfaceView);
        surfaceHolder = surfaceView.getHolder();
        surfaceHolder.addCallback(this);
        surfaceHolder.setType(SurfaceHolder.
                SURFACE_TYPE_PUSH_BUFFERS);
        camera = Camera.open();
        btn1.setOnClickListener(new OnClickListener() {
                @Override
                public void onClick(View v) {
                        camera.startPreview();
                }
        });
}
@Override
public void surfaceCreated(SurfaceHolder holder) {
        try {
                camera.setPreviewDisplay(surfaceHolder);
        } catch (final Exception e) {
        }
}
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
        camera.stopPreview();
        camera.release();
        camera = null;
}
```

onCreate part of Acitivity lificyle
Called when user starts apps
Camera acquired
Invoked when user clicks Button 1
Camera usage started captured video is shown on screen
Preview stopped
Camera released
Entry Node (onCreate called)
Btn Player1
Btn Player2
Btn Pause
Exit Node (onPause called)
Where the wakelock is needed
Where the wakelock is acquired

Figure 4.10: Java code fragment showing non-optimal use of camera resource acquisition, described by [16]. The wake lock[a] is acquired from the Entry Node to the Exit Node but it's actually necessary only during a piece of the sequence.

[a]Wake locks are power-managing software mechanisms used to indicate the device to stay on.

techniques, authors use XEEMU[97] tool which is a software power estimation tool supporting C/C++ based estimation. They prepared example source codes based on sample codes used in M. Flower's studies [66] and customized them to be executed without any external input as they can cause different power consumption estimation for the same code. Their evaluation shows that 30 refactoring techniques cause energy consumption regressions while 33 others bring no change or are even energy-efficiencies. Through this paper, they demonstrate that code refactoring process could improve energy consumption of applications, in the same way as their primary goal which is improving maintainability of software.

Related work [64] defines the conditions of power waste for mobile application. Based on his prior work [65], they were able to create a code refactoring technique to respond to the waste conditions. They also demonstrate that their solution is appropriate for reducing power consumption of refactored code. Nevertheless, the detail refactoring process is not fully described in his work [64], preventing any other use of his technique.

The compiler is also an integral part of the development process, therefore it can also play a role in reducing the energy consumption of the application. Yet there is very few work on the subject. X. Chen and al [26] have developed a solution called Android Energy Profiler (AEP) to analyze the energy impact of programming languages such as C, C++ or Java. This analysis also proposes programming styles with appropriate use of native code[12] in order to optimize the way the application will be compiled. This work cannot however be exploited

[12]Native code is a programming code compiled to run with a particular processor and its set of instructions.

Algorithm 1 Re-factoring design expression

```
1: Input:
2: $App$: App source files
3: $AppCt$: Category such as Games, Travel, Books, etc
4: Output:
5: $Ref_{app}$: Re-factored design expression
6:
7: $Des_{app} \leftarrow$ GenerateDesign($App$)
8: $\langle v, r \rangle \leftarrow$ CheckViolation($Des_{app}, App, AppCt$)
9: while $v \in V$ do
10:     $\langle X_r, U_r, Y_r \rangle \leftarrow$ GetResSymbols($Des_{app}, r$)
11:     if ($v = SubOptimalBinding$) then
12:         $Ref_{app} \leftarrow$ insertBefore($Des_{app}, U_r, X_r$)
13:         $Ref_{app} \leftarrow$ insertAfter($Ref_{app}, U_r, Y_r$)
14:     end if
15:     if ($v = NestedUsage$) then
16:         $Ref_{app} \leftarrow$merge($Des_{app}, X_r$)
17:         $Ref_{app} \leftarrow$merge($Ref_{app}, Y_r$)
18:     end if
19:     if ($v = TradeOff$) then
20:         $Ref_{app} \leftarrow$ reconfigure($Des_{app}, AppCt, X_r$)
21:     end if
22:     if ($v = ResourceLeak$) then
23:         $Ref_{app} \leftarrow$
24:         insertAfter($Des_{app}, U_r$, {getRelSysCall($r$)})
25:     end if
26:     $\langle v, r \rangle \leftarrow$ getDefect($Res_{app}, App, AppCt$)
27: end while
```

Figure 4.11: Algorithm that computes design-expressions to implement energy-efficiency guidelines. Decisions are made based on following rules: Sub-optimal binding, Nested usage, Tradeoff and resourceleak. [16]

all along the application development but it is rather there to guide the choices of implementation in the first phases of design.

Some researchers [16] are also recently proposing a solution that re-factor the source code to make it energy-efficient. They say this technique is capable of automatically refactor android applications in order to assist in energy awareness during application development. In this approach, they developed a framework that comprises a set of energy-efficiency guidelines to refactor the design-expression of an application. This design-expression is a regular expression that represents the modules that are energy-intensive and operate smartphone resources. Their framework detects and refactors instances of energy-drain within the target application using a refactoring design expression algorithm (Figure 4.11). For example, one of the guideline-based refactoring is to ensure that resources usage is acquired as late as possible and released as soon as possible. The Figure 4.10 is an example showing that this their rules bring energy improvement on common design patterns. They demonstrate that such technique can achieve up to 29

% of application energy-saving. The framework can be executed on both under development or released applications. Such energy-aware design can also raise the programmer's consciousness for future developments. Moreover, as the changes are made to the application source code and not onto the platform, the behaviour should react similar, independently on the platform and the device. However, the energy-efficiency guidelines do not cover all case as they say, the features of real-life applications may vary widely and their insights are not necessarily shared with the developer community. Also the guidelines may not be applicable, depending on the context in which a feature is used and what is the purpose of using it in this way.

Chapter 5

Conclusion

The introduction of smartphones has brought its share of new challenges that need to be addressed, such as saving smartphone's energy. Batteries with limited autonomy are no longer sufficient to run all the applications used on a daily basis. Applications rely on hardware resources to provide features for the user's need. Those demands have an energy cost that drains the battery's autonomy. It becomes necessary to efficiently use the electronic components. Meanwhile, researchers are studying the possibilities to estimate, analyze and optimize resource usage in order to make more energy-efficient applications. Large companies are also concerned by the energy saving problematic and are currently forming research groups (e.g. IBM and Intel forming GreenIT [88]) to carry out studies in this field. In spite of the efforts, available tools that assist developer during the mobile application design are still few in number, they require a large investment of time and money or are not very accurate. Unlike standards that enforce application's security or general performance of the smartphone, there is no common application energy-efficiency guideline that programmers can rely on.

In this thesis we attempted to answer the question of how software engineering can resolve the application power consumption problematic using program analysis. This work describes and analyzes the main factors of energy consumption of the resources in a smartphone, the existing solutions that estimate, analyze and report this energy consumption and the actual solutions to optimize resource usage in mobile applications.

We first tried to understand the main reasons for which electronic components consume energy. They allow mathematical operations to be performed and the information to be stored, communicated, displayed and captured on the smartphone such as audio, video and positioning data. Following this analysis, we understand that the technologies and features used inside the smartphone have a significant impact on smartphone's power consumption. However, the hardware designers of components are aware of the need to reduce this energy consumption. In fact, hardware technologies evolve and tend towards less energy-consuming solutions while increasing performance and thinness.

Secondly, we reviewed the techniques for estimating and analysing the energy consumption of mobile applications and consequently the cost of using resources.

The most suitable solutions for estimation are the constructions of power models. They provide data that are essential for software application designers. Model's inputs can be related to the use of smartphones, to the cost of a method call or instruction and the cost of using interfaces to hardware components. It is clear that if we want a fine-grained estimation, power modeling must allow as much information as possible. Instruction-based modeling techniques seem to have a promising future in this area where the energy cost of executing a line of source code can be estimated. In addition, some work has showing that that it is possible to integrate power consumption profiling tools into integrated development environment (IDE) currently used by programmers. This would make it possible to quickly and effortlessly analyze the energy cost of implemented designs. In that chapter, we have also looked at solutions that support software developers and more particularly that raise awareness of best practices in the community. Techniques such as mining repositories or energy efficiency repositories are proposed and could meet this need of sharing information about design energy cost. However, the existing solutions need the support of the community as their relevance lies in generalizing the use of tools to build a sufficient data set.

Finally, we evaluated techniques for optimizing the energy consumption of mobile applications. Many solutions can be found at several levels between the resource and the application, but we selected the ones that can improve the management of the most demanding resources and those that provide an alternative optimization solution for applications with a finished design. In this approach, we evaluated solutions that improve the power management of the computing units as well as the display units. Most techniques perform voltage and/or frequency regulation on memory, processing unit, graphical unit and displays. Researchers asked themselves if the resource actually needs all this power for the duration of its operation. Their study answers the question by showing that alternatives exist and they can maintain initial performance and usability. However, the techniques mentioned in the chapter require specific technologies and a lot of empirical analysis. We hope that electronic components manufacturers are going to continue the improvements in reducing the product's energy consumption. The last area we reviewed deals with task offloading techniques and code refactoring techniques. The recent solutions we evaluated have shown a promising future as they demonstrate the existing techniques to offload smartphone's resources from the weight of carrying out energy-intensive tasks. The energy consuming tasks can be executed on remote servers provided with powerful hardware to perform operations required by the smartphone. The last two years have brought a lot of progress with solutions to limit changes in the source code and in network quality analysis, and more efficient task decision algorithms whose smartphone's external execution is worth it. This section reviews refactoring techniques use for optimizing the design code of the application. Based on pre-established rules, researchers show that the source code can be maintained while being the energy-efficient. Most of the solutions can automatically refactor application code with reliability. However, the question of whether it is possible to trust pre-established rules that do not necessarily take the context of use into account arises.

In conclusion, we believe that there is an urgent need to raise energy-saving awareness among the software development community. Nonetheless, on the researchers' side, increasingly effective analyzing tools are already proposed. The best analyzing techniques yet are those with the capability to analyze and estimate energy consumption at fine-grained level. However, not many are fully extensible to different platforms than studied cases. Today it is still possible to collect power consumption data and it is now time to find means that will enable to share and benefit the developers' community of those markers. In regard to application legacy code, alternatives to optimize applications with few code modification are emerging. The latter solutions would help to constraint application's energy-efficiency as a common software quality attribute. In this approach, the motivation of the software developers is a key element therefore, the tools' usability and reliability shall be the main aspects to go through (e.g integration in software development toolkit).

Bibliography

[1] A. Gupta, T. Zimmermann, C. Bird, N. Naggapan, T. Bhat, and S. Emran, *Energy Consumption in Windows Phone*, Microsoft Research, Tech, (2011).

[2] ——, *Energy Consumption in Windows Phone*, Microsoft Research, Tech. Rep. MSR-TR-2011-106, (2011).

[3] A. Hindle, *Green Mining: A Methodology of Relating Software Change to Power Consumption*, in MSR, (2012), pp. 78 – 87.

[4] ——, *Green mining: Investigating power consumption across versions*, in MSR, (2012), pp. 1301–1304.

[5] A. Hindle, A. Wilson, K. Rasmussen, J. Barlow, J. Campbell, and S. Romansky, *GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework*, in Mining Software Repositories (MSR), 2014 11th IEEE Working Conference on. ACM, (2014).

[6] A. Mukherjee and T. Chantem, *Energy management of applications with varying resource usage on smartphones*, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 37, no. 11, (2018), pp. 2416 – 2427.

[7] A. Noureddine, A. Bourdon, R. Rouvoy, L. Seinturier, *A Preliminary Study of the Impact of Software Engineering on GreenIT*, First International Workshop on Green and Sustainable Software, GREENS 2012, Zurich, Switzerland, (2012), pp. 21 – 27.

[8] A. Pathak, Y. C. Hu, and M. Zhang, *Where is the Energy Spent inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof*, Proceedings of the 7th ACM european conference on Computer Systems, ser. EuroSys '12, (2012), pp. 29 – 42.

[9] A. Pathak, Y. Hu, M. Zhang, P. Bahl, and Y. Wang, *Fine-Grained Power Modeling for Smartphones using System Call Tracing*, EuroSys '11 Proceedings of the sixth conference on Computer systems, (2011), pp. 153 – 168.

[10] Adrian, *Understanding the Scan Modes in WiFi Explorer Pro*, www.adriangranados.com/blog/understanding-scan-modes-wifiexplorerpro, (2017).

[11] Android OS Author, *CPU Governors, Hotplug drivers and GPU governors*, https://androidmodguide.blogspot.com/p/blog-page.html, (2017).

[12] APKTool, https://code.google.com/p/android-apktool/, (2017).

[13] Apple Inc., *iOS Application Programming Guide: Tuning for Performance and Responsiveness*, http://ur1.ca/696vh, (2010).

[14] J. S. P. G. K. A. L. A. M. C. C. B. Anand, K. Thirugnanam and R. K. Balan, *Adaptive display power management for mobile games*, in Proc. 9th Int. Conf. Mobile Syst., Appl., Services (MobiSys), (2011), pp. 57–70.

[15] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, *CloneCloud: elastic execution between mobile device and cloud*, EuroSys '11 Proceedings of the sixth conference on Computer systems, (2011), pp. 301 – 314.

[16] A. Banerjee and A. Roychoudhury, *Automated re-factoring of android apps to enhance energy-efficiency*, 2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), (2016), pp. 139–150.

[17] BCEL library, http://commons.apache.org/bcel/, (2017).

[18] Blending - OpenGL, https://www.khronos.org/opengl/wiki/Blending, (2017).

[19] Bluetooth - Wikipedia, https://en.wikipedia.org/wiki/Bluetooth, (2020).

[20] C. Gray, *Performance Considerations for Windows Phone 7*, http://create.msdn.com/downloads/?id=636, (2010).

[21] C. Seo, S. Malek, N. Medvidovic, *An Energy Consumption Framework for Distributed Java-Based Systems*, in ASE '07, (2007), pp. 421 – 424.

[22] ——, *Component-level energy consumption estimation for distributed java-based software systems*, in Component-Based Software Engineering Springer, (2008), pp. 97 – 113.

[23] A. Carroll, *Understanding and reducing smartphone energy consumption*, PHD Univ. New South Wales, Sydney, NSW, Australia, (2017).

[24] H. Chen, Y. Li, and W. Shi, *Fine-grained power management using process-level profiling*, Sustainable Computing: Informatics and Systems, vol. 2, no. 1, (2012), pp. 33 – 42.

[25] X. Chen, Y. Chen, and Chun Jason Xue, *Datum: Dynamic tone mapping technique for oled display power saving based on video classification*, 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), (2015), pp. 1–6.

[26] X. Chen and Z. Zong, *Android app energy efficiency: The impact of language, runtime, compiler, and implementation*, in Proc. IEEE Int. Conf. Big Data Cloud Comput. (BDCloud), Social Comput. Netw. (SocialCom), Sustain. Comput. Commun. (SustainCom),, (2016), pp. 485 – 492.

[27] M. Conti, B. Crispo, D. Diodati, J. Nurminen, C. Pinotti, and T. Teemaa, *Leveraging parallel communications for minimizing energy consumption on smartphones*, IEEE Transactions on Parallel and Distributed Systems, vol 26, (2014).

[28] D. Brooks, V. Tiwari, and M. Martonosi, *Wattch: a framework for architectural-level power analysis and optimizations*, In ACM SIGARCH Computer Architecture News, (2000), pp. volume 28, 83–94.

[29] D. Burger and T. M. Austin, *The SimpleScalar Tool Set, Version 2.0*, Computer Architecture News, (1997), pp. 13 – 25.

[30] D. Li, S. Hao, W. G. Halfond, and R. Govindan, *Calculating Source Line Level Energy Information for Android Applications*, in Proceedings of the 2013 International Symposium on Software Testing and Analysis, (2013), pp. 78 – 89.

[31] N. C. D. Shin, Y. Kim and M. Pedram, *Dynamic voltage scaling of oled displays*, in Proc. 48th Design Automat. Conf. (DAC), (2011), pp. 53–58.

[32] Dalvik Software - Wikipedia, https://en.wikipedia.org/wiki/Dalvik$_(software)$, (2017).

[33] A. C. de Melo, *The new linux 'perf' tools*, in Slides from Linux Kongress, vol. 18, (2010).

[34] dex2jar tool, http://code.google.com/p/dex2jar/, (2017).

[35] O. Djedidi, M. A. Djeziri, N. K. M'Sirdi, and A. Naamane, *A novel easy-to-construct power model for embedded and mobile systems - using recursive neural nets to estimate power consumption of arm-based embedded systems and mobile devices*, (2018).

[36] Do, T., Rawshdeh, S., Shi, W., *pTop: A Process-level Power Profiling Tool*, In Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower), (2009), pp. 762 – 767.

[37] Dynamic Voltage Scaling - Wikipedia, https://en.wikipedia.org/wiki/Dynamic$_voltage_scaling$, (2020).

[38] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, *MAUI: Making smartphones last longer with code offload*, In MobiSys, (2010).

[39] Erik Mouv, *Linux Kernel Procfs Guide*, http://www.compsoc.man.ac.uk/ moz/kernelnewbies/documents/kdoc/2.5/procfs-guide.pdf, (2001).

[40] P. K. D. P. et al., *Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage*, IEEE Access, vol. 7, (2019), pp. 182113–182172.

[41] Fontoura Cupertino, Leandro and Da Costa, Georges and Sayah, Amal and Pierson, Jean-Marc, *Valgreen: an Application's Energy Profiler*, International Journal of Soft Computing and Software Engineering, vol. 3, (2013), pp. 549 – 556.

[42] G. Singh and M. K. Rohit and C. Kumar and K. Naik, *MultiDroid: An Energy Optimization Technique for Multi-Window Operations on OLED Smartphones*, IEEE Access, 6 (2018), pp. 31983–31995.

[43] Generate CPU, Memory and I/O report using SAR command, www.linuxtechi.com/generate-cpu-memory-io-report-sar-command, (2020).

[44] H. Jiang, H. Yang, S. Qin, Z. Su, J. Zhang, J. Yan, *Detecting Energy Bugs in Android Apps Using Static Analysis*, In: Duan Z., Ong L. (eds.) Formal Methods and Software Engineering. ICFEM 2017, (2017), pp. Vol. 10610: 192 – 208.

[45] Haowei Wu, Shengqian Yang and Atanas Rountev, *Static Detection of Energy Defect Patterns in Android Applications*, Proceedings of the 25th International Conference on Compiler Construction, ACM, (2016), pp. 185 – 195.

[46] Hidden Markov Model - Wikipedia, en.wikipedia.org/wiki/Hidden$_M arkov_m odel$, (2020).

[47] Hoffmann J, Ussath M, Holz T, and al, *Slicing droids: program slicing for smali code*, Automated Software Engineering (ASE), (2013), pp. 1844 – 1851.

[48] M. Hoque, *Modeling, profiling, and debugging the energy consumption of mobile devices*, ACM Computing Surveys, (2015).

[49] IBM, *IBM Active Energy Manager*, http://www.ibm.com/systems/management/director/abo
(2011).

[50] IEEE 802.11 - Wikipedia, fr.wikipedia.org/wiki/IEEE$_8$02.11, (2020).

[51] Intel, *LessWatts.org - Saving Power on Intel systems with Linux*, http://www.lesswatts.org, (2011).

[52] IPC - Wikipedia, https://en.wikipedia.org/wiki/Instructions$_per_cycle$, (2020).

[53] J. Flinn and M. Satyanarayanan, *PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications*, In Second IEEE Workshop on Mobile Computing Systems and Applications, (1999), pp. 2 – 10.

[54] J. J. Park, J.-E. Hong, *An Approach to improve software safety by Code refactoring*, Proc. of Korea Computer Congress, (2013), pp. 532 – 534.

[55] J. J. Park, J.-E. Hong, S.-H. Lee, *Investigation for Software Power Consumption of Code Refactoring Techniques*, SEKE, (2014), p. Vol. 3.

[56] K. Aggarwal, C. Zhang, J. C. Campbell, A. Hindle, E. Stroulia, *The power of system call traces: Predicting the software energy consumption impact of changes*, Proceedings of 24th Annual International Conference on Computer Science and Software Engineering, (2014), pp. 219 – 233.

[57] K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, *The energy/frequency convexity rule: Modeling and experimental validation on mobile devices*, in Proc. Int. Conf. Parallel Process. Appl. Math. (PPAM), (2014).

[58] 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 Codesign and System Synthesis (CODES+ISSS), Scottsdale, AZ, USA, (2010), pp. 762 – 767.

[59] S. Li and S. Mishra, *Optimizing power consumption in multicore smartphone*, J. Parallel Distrib. Comput., vol. 95, (2016), pp. 124 – 137.

[60] Lint, http://tools.android.com/tips/lint, (2017).

[61] Linux - Wikipedia, https://en.wikipedia.org/wiki/Linux, (2020).

[62] Y. Liu, C. Xu, and S. Cheung, *Where has my battery gone? finding sensor related energy black holes in smartphone applications*, (2013).

[63] M. Dong and L. Zhong, *Sesame: Self-Constructive System Energy Modeling for Battery-Powered Mobile Systems*, In Proc. of MobiSys, (2011), pp. 335 – 348.

[64] M. Gottschalk, J. Jelschen, and A. Winter, *Energy-Efficient Code by Refactoring*, 15. Workshop Software-Reengineering, (2013).

[65] M. Gottschalk, M. Josefiok, J. Jelschen, A. Winter, *Removing Energy Code Smells with Reengineering Services*, GI-Jahrestagung, (2012), pp. 441 – 455.

[66] Martin Fowler, *Refactoring: improving the design of existing code*, Addison-Wesley Professional, (1999).

[67] MonkeyRunner, http://developer.android.com/guide/developing/tools/monkeyrunner, (2017).

[68] I. Moura, G. Pinto, F. Ebert, and F. Castor, *Mining energy-aware commits*, 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, (2015), pp. 56 – 67.

[69] D. J. Muñoz Guerra, J. Montenegro, M. Pinto, and L. Fuentes, *Green security plugin for pervasive computing using the hadas toolkit*, (2017), pp. 796–803.

[70] D. J. Muñoz Guerra, M. Pinto, and L. Fuentes, *Green software development and research with the hadas toolkit*, (2017), pp. 205–211.

[71] ——, *Hadas and web services: Eco-efficiency assistant and repository use case evaluation*, (2017), pp. 1–6.

[72] ——, *Hadas: analysing quality attributes of software configurations*, (2019), pp. 1–4.

[73] Muñoz Guerra, Daniel Jesus, *Achieving energy efficiency using a Software Product Line Approach*, (2017), pp. 131–138.

[74] H. L. D. P. T. C. N. D. Lane, E. Miluzzo and A. T. Campbell, *A survey of mobile phone sensing*, IEEE Commun. Mag., vol. 48, no. 9, (2010), pp. 140 – 150.

[75] J. L. D. Neto, S. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, *Uloof: A user level online offloading framework for mobile edge computing*, IEEE Transactions on Mobile Computing, 17 (2018), pp. 2660–2674.

[76] Nikon Technology", *Liquid-crystal display (LCD) and organic LED (OLED) display*, www.nikon.com/products/fpd/technology/story01.html, (2020).

[77] NVIDIA Corporation. Nvidia Tegra - The Mobile Super Chip., http://www.nvidia.com/object/tegra.html, (2020).

[78] OpenGL - Wikipedia, https://fr.wikipedia.org/wiki/OpenGL, (2017).

[79] P. Peterson, D. Singh, W. Kaiser, and P. Reiher, *Investigating energy and security trade-offs in the classroom with the atom LEAP testbed*, 4th Workshop on Cyber Security Experimentation and Test (CSET), (2011), p. 11.

[80] Patrick Mochel, *The sysfs Filesystem*, http://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf, (2005?).

[81] Pohua P. Chang., *Trace selection for compiling large C application programs to microcode*, In 21th AnnualWorkshop on Microprogramming and Microarchitecture (MICRO 21), (1988), pp. 21 – 29.

[82] M. B. R. Duan and C. Gniady, *Exploring memory energy optimizations in smartphones*, 011 International Green Computing Conference and Workshops, (2011), pp. 1 – 8.

[83] S. H. A. H. M. S. A. I. A. A. S. A. M. K. S. R. W. Ahmad, A. Gani and J. J. Rodrigues, *A survey on energy estimation and power modeling schemes for smartphone applications*, Int. J. Commun. Syst., vol. 30, no. 11, (2017).

[84] R.W. Stevens and S. A. Rago, *Advanced Programming in the UNIX(R) Environment (2nd Edition)*, Addison-Wesley Professional, (2005).

[85] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, *Estimating Android Applications' CPU Energy Usage via Bytecode Profiling*, First International Workshop on Green and Sustainable Software (GREENS), (2012), pp. 1 – 7.

[86] ——, *Estimating Mobile Application Energy Consumption Using Program Analysis*, Proceedings of the 2013 International Conference on Software Engineering, (2013), pp. 92 – 101.

[87] N. F. K. P. A. H. J. P. S. M. A. H. Jafri, O. Ozba and H. Tenhunen, *Architecture and implementation of dynamic parallelism, voltage and frequency scaling (pvfs) on cgras*, ACM J. Emerg. Technol. Comput. Syst., vol. 11, no. 4, (2015), p. Art. no. 40.

[88] S. Murugesan, *Harnessing Green IT: Principles and Practices*, IT Professional, (2008), pp. vol. 10, no. 1, pp. 24–33.

[89] Samdharsi Kumar, *What is the difference between WiFi authentication and association?*, www.quora.com/What-is-the-difference-between-WiFi-authentication-and-association, (2016).

[90] T. Ball and J. Larus, *Efficient Path Profiling*, MICRO 29, (1996), pp. 46 – 67.

[91] Tianyong Wu, Jierui Liu, Xi Deng, Jun Yan and Jian Zhang, *Relda2: an effective static analysis tool for resource leak detection in Android apps*, ASE2016, (2016), pp. 762 – 767.

[92] TianyongWu, Jierui Liu, Zhenbo Xu, Chaorong Guo, Yanli Zhang, Jun Yan and Jian Zhang, *Light-Weight, Inter-Procedural and Callback-Aware Resource Leak Detection for Android Apps*, IEEE Trans. Software Eng, (2016), pp. 1054 – 1076.

[93] Wi-Fi - Wikipedia, https://en.wikipedia.org/wiki/Wi-Fi, (2020).

[94] C. Xiaomeng, N. Ding, A. Jindal, Y. Hu, M. Gupta, and R. Vannithamby, *Smartphone energy drain in the wild*, ACM SIGMETRICS Performance Evaluation Review, 43 (2015), pp. 151–164.

[95] H. Xie, H. Tang, and Y.-H. Liao, *Time series prediction based on narx neural networks: An advanced approach*, Proceedings of the 2009 International Conference on Machine Learning and Cybernetics, 3 (2009), pp. 1275 – 1279.

[96] Y. Kwon, Z. Lee, Y. Park, *Performance-based Refactoring: Identifying & Extracting Move-method Region*, Journal of KIISE: Software and Applications, (2013), pp. 567 – 574.

[97] Z. Herczeg, D. Schmidt, A. Kiss, N. Wehn, T. Gyimothy, *Energy simulation of embedded XScale systems with XEEMU*, Journal of Embedded Computing, (2009), pp. 209 – 219.

Glossary

AC Alternative Current. 35

ACPI Advanced Configuration and Power Interface. 25

AEP Android Energy Profiler. 50

AGPS Assisted Global Positioning System. 21

AM Amplitude Modulator. 44

AMOLED Active-Matrix Organic Light-Emitting Diode. 44

API Application Programming Interface. 31

CAM Content Addressable Memory. 24

CFG Call Graph Flow. 23

DTS Delay-Tolerant Services. 41

DVFS Dynamic and Voltage Frequency Scaling. 40

DVS Dynamic Voltage Scaling. 44

FM Frequency Modulation. 20

FSM Finite State Machine. 26, 31

GPS Global Positioning System. 19

GPU Graphical Processing Unit. 14

HMM Hidden Markov Model. 44

IPC Instruction Per Cycle. 40

JFP Java PathFinder. 34

JVM Java Virtual Machine. 27

LCD Liquid Crystal Display. 16, 42

LED Light-Emitting Diode. 16

LUT Look Up Table. 40

MCC Mobile Cloud Computing. 46

MOSFET Metal Oxide Semiconductor Field Effect Transistor. 15

MSR Mining Software Repository. 35

OLED Organic Light-Emitting Diode. 42

OS Operating System. 27

PAVM Power-Aware Virtual Memory. 41

PCM Phase Change Memory. 41

PID Process IDentifier. 27

POC Proof Of Concept. 44

PWM Pulse Width Modulator. 44

RAM Random Access Memory. 15

RTS Real-Time Services. 41

SADD Static Application Analysis Detector. 33

SAR System Activity Report. 35

SEEP Software Environment Energy Profile. 28

SoC System on a Chip. 23

SPL Software Product Line. 37

TDL Time Delay Line. 28

UHF Ultra High Frequency. 18

ULOOF User Level Online Offloading Framework for Mobile Edge Computing. 47

VM Virtual Machine. 47