Mobile Netw Appl
DOI 10.1007/s11036-013-0470-y

Adaptive and Flexible Smartphone Power Modeling

**A. A. Nacci · F. Trovò · F. Maggi · M. Ferroni ·
A. Cazzola · D. Sciuto · M. D. Santambrogio**

Abstract Mobile devices have become the main interaction mean between users and the surrounding environment. An indirect measure of this trend is the increasing amount of security threats against mobile devices, which in turn created a demand for protection tools. Protection tools, unfortunately, add an additional burden for the smartphone's battery power, which is a precious resource. This observation motivates the need for smarter (security) applications, designed and capable of running within adaptive energy goals. Although this problem affects other areas, in the security area this research direction is referred to as "green security". In general, a fundamental need to the researches toward creating energy-aware applications, consist in having appropriate power models that capture the full dynamic of devices and users. This is not an easy task because of the highly dynamic environment and usage habits. In practice, this goal requires easy mechanisms to measure the power consumption and approaches to create accurate models. The existing approaches that tackle this problem are either not accurate or not applicable in practice due to their limiting requirements. We propose MPower, a power-sensing platform and adaptive power modeling platform for Android mobile devices. The MPower approach creates an adequate and precise knowledge base of the power "behavior" of several different devices and users, which allows us to create better device-centric power models that considers the main hardware components and how they contributed to the overall power consumption. In this paper we consolidate our perspective work on MPower by providing the implementation details and evaluation on 278 users and about 22.5 million power-related data. Also, we explain how MPower is useful in those scenarios where low-power, unobtrusive, accurate power modeling is necessary (e.g., green security applications).

Keywords Adaptive power modeling · Security applications · Smart power management · Data collection · Mobile devices

A. A. Nacci · F. Trovò · F. Maggi · M. Ferroni · A. Cazzola · D. Sciuto · M. D. Santambrogio (✉)
Politecnico di Milano, Milano, Italy
e-mail: marco.santambrogio@polimi.it

A. A. Nacci
e-mail: nacci@elet.polimi.it

F. Trovò
e-mail: trovo@elet.polimi.it

F. Maggi
e-mail: maggi@elet.polimi.it

M. Ferroni
e-mail: matteo.ferroni@mail.polimi.it

A. Cazzola
e-mail: andrea.cazzola@mail.polimi.it

D. Sciuto
e-mail: sciuto@elet.polimi.it

1 Introduction

Mobile devices, in particular smartphones and tablets, have become the device of choice for many users, and the market is growing at unprecedented rates. As an indicator, the mobile network traffic as a percentage of the global Internet traffic has an estimated growing rate of 1.5× per year.[1] Given the popularity of smartphones [21], an increasing amount of security threats have begun targeting such devices. As modern smartphones are often used

[1] http://www.kpcb.com/insights/2013-internet-trends

to store user-centric sensitive information (e.g., contacts, credentials) or, worse, to perform financial transactions, they have become an appealing target for cybercriminals. Indeed, between 2010 and 2012, about [25] 50 families of malicious applications have been found to target the Android system. In 2012–2013, this phenomenon exploded and, as of June 2013, several thousands of variants of known and new families of Android malware are found every day. On the one hand, this created a demand for protection tools, to which security vendors responded with about 80 anti-malware applications for Android, free or paid, distributed through the Google Play Store. On the other hand, installing and running another application translates into an increased burden for the smartphone's battery power, which is a precious resource as we noticed in [19]. Indeed, checking the storage for the presence of malicious applications or, for instance, screening incoming text messages for malicious links is costly in terms of both power and speed.

The above considerations naturally motivate the need for smarter security solutions, capable of running and performing effectively under an energy-aware perspective [5]. This research direction, often referred to as "green security", aims at embedding modern smartphone's power constraints, models and usage patterns during the design of security solutions (e.g., anti-malware applications). For instance, it would be desirable to have a tool capable of easily profiling a device, used by a certain user, with and without a security solution installed, so as to evaluate the modification on the device dynamic caused by such a security solution. Another related research direction that received some attention consists in capturing the power model of a threat (e.g., running malicious application) to create so-called "power signatures" [16], which can be leveraged to detect the presence of such threat. Similarly, other researchers have proposed to detect threats under the assumption that they introduce a measurable deviation from the user's typical power usage pattern. However, such assumption have been recently been showed to be quite hard to ensure [11], at least for a limited number of devices. Indeed, the highly-dynamic conditions under which a smartphone is used today (e.g., changing habits, new applications installed, removed, or updated) make it very difficult to create robust power models and, more importantly, to detect variations accurately.

Given these premises, we conclude that a fundamental need common to both the aforementioned research directions, as well as other research fields, consists in having appropriate power models. These models must capture the full dynamic of both the device and the user, which is not an easy task because of the dynamic environment and changing conditions of modern smartphones. In practice, this requires mechanisms to measure the power consumption. These mechanisms are also challenging to design and develop, because of their conflicting requirements. First, they must not introduce a significant power load themselves. Second, they should gather precise measurements. Third, they must run on the mobile device itself—without substantial modifications. For example, hardware-based approaches (e.g., employing a sophisticated sensor attached to the device in a laboratory) are not desirable because they constrain the device to a fixed location, changing completely its usage model (although they guarantee high-precision measurement).

As discussed in Section 2.2, the existing approaches that tackle these problems are either not accurate (e.g., too user centric) or not applicable in practice due to their limiting requirements (e.g., hardware based). We first highlighted the need for better approaches in [3], where we presented the concept of MPower, an adaptive power-modeling system—and Android application—capable of non-obtrusive power measurements and modeling. Approaches (and apps) that does both measurement and modeling on the device introduce an excessive power load, mainly due to the power model computation. As we discussed in [8], on the one hand an accurate power model requires expensive computation, on the other hand, the short battery life makes this unfeasible in practice.

In this paper, we consolidate the aforementioned perspective work by providing the implementation details and evaluation of MPower. Also, we explain how MPower is useful in those scenarios where low-power, unobtrusive, accurate power modeling is necessary (e.g., green security applications). The design of MPower is not obtrusive in two ways: first, it does not require any software or hardware modifications on the device; second, it offloads expensive computation to remote servers, keeping only a lightweight power-sensing process on the smartphone. Differently from previous work, MPower does not change the way the device is used (i.e., does not require obtrusive hardware modifications): this ensures a decent user base, which in turns means that a good variety of devices are considered by the power model. Because of its peculiar characteristics and flexibility, MPower has a very broad application domain, including the aforementioned (green) security research direction.

MPower is both a power-sensing Android application—distributed through the Google Play Store and described in Section 3—and power-modeling approach (described in Section 4). Once installed, the MPower app starts with a data-collection phase on the device (described in Section 3.1), which logs time series of network utilization, battery charge level, etc. This data is sent to a remote server where a pre-built power model is tuned according to the user-specific behavior. This procedure creates an adequate and precise knowledge base of the power "behavior" of several different devices and users, which allows MPower to create better device-centric power model, which considers the main hardware components and how they contributed to the overall power consumption. In turn, this approach provides the users with tailored power models, toward better power-management strategies. As detailed in Section 4, we rely on machine-learning techniques to generate the power models.

2 Background and state of the art

Although mobile devices have evolved from simple devices to complex systems, they still belong to a constrained and fluctuating environment: *constrained* because resources are limited, both in terms of performance and power; *fluctuating* because both the external environment and the resources availability can greatly vary over time. The battery energy is an example of a resource that influences the whole system functioning and thus must be modeled carefully. This modeling must take the constrained and fluctuating environment into account: In other words, the power model should be adaptive.

The remainder of this section contextualizes the problem of power modeling on mobile devices with a focus on the Android platform (Section 2.1). We describe what data can be measured from the environment and what data about power consumption can be collected through the Android APIs. Last, in Section 2.2, we overview the relevant work in the area and discuss the respective strengths and weaknesses, which motivate our research.

2.1 Android as a sensing platform

Modern smartphones are equipped with several sensors, which can be used to sense both internal (e.g., battery level, CPU load) and external (e.g., movement, location) device conditions. The Android SDK includes a set of APIs that ensure easy access to the available sensors and observe the device context. For the purpose of this work we need access to the battery, location and network APIs, which we briefly overview in the reminder of this section.

Information about the battery status is often coarse grained, because most devices do not include a physical measure about electrical current, or more precise hardware sensors. The only information exposed by the system to the developers are the battery percentage, voltage, temperature and an indicator of the battery health status. Also, Android offers a great variety of sensors that can be used to sense the external environment [15]. The device geolocation [2] is ensured by the *Location framework*, which reads data from different sources. The GPS is the most accurate source, it works well outdoors and is highly battery hungry. If an accurate position is not mandatory, the location can also be determined using cell tower or Wi-Fi information, which consume less battery power. Wireless technologies such as Bluetooth and NFC are also supported by Android. These can be used, for instance, to discover nearby devices in the surrounding environment. Some other APIs are provided to scan for other devices, query for paired devices and exchange data between paired devices.

The `SensorManager` class manages the access to the hardware sensors available on the device, which may include accelerometers, magnetometers, pressure and temperature sensors. Each sensor data is identified through a type, a sampling rate and an accuracy. The touch screen itself is recognized as a sensor. As a best practice, developers are invited to use sensors with care, because most of them require a non-negligible amount of power. The sampling frequency is particularly important in this regard: Choosing the “right” sampling period requires empirical work, to ensure that an application is able to gather enough data for its purposes, without influencing the behavior of the system.

MPower targets the Android platform because of its popularity and open design, which makes it a great research tool. Indeed, the large amount of data that can be directly retrieved from or inferred by an Android device makes its a good choice for many applications such as MPower.

Android does not provide development tools to analyze how much an application, a component or the whole system is consuming. Since version 2.3, Android provides statistics to the end user regarding the battery usage of each application and component. In practice, a screen lists all the applications and services that have consumed at least the 2 % of the battery life since last battery recharge. Also, a graph shows the battery discharging over time along with some metadata for each listed application. Unfortunately, this data cannot be accessed programmatically, as no API is exposed. In this regard, MPower can be seen as an effort to mitigate this limitation by showing that such statistics can be computed from raw data in the application scope.

2.2 Related work

Power-energy models differ in the method to generate the model, measurement methodology, granularity, adaptability to changes (e.g., user-device behavior), and target devices. In this regard, several methodologies to create power models that combine together all the aforementioned properties can be found in the literature. The generation of a system-level power-consumption model can be performed at runtime or offline: One the one hand, a runtime-generated model is able to adapt to new (software) updates or even new devices, without the need to (re)generate the whole model from scratch; on the other hand, an adaptive model generation usually relies only on software APIs to gather information from the actual battery state, information that cannot be as precise as it would be using an external measurement system. Another important difference among the proposed approaches is related to the granularity of the generated model: Some works try to develop a model able to compute the entire system power consumption at a given time. Clearly, offline models have symmetric properties (i.e., precise measurement yet static model estimation). Other works are related to model a single application power consumption. Finally, some works take into account also the user experience and how different device usage patterns can influence the overall power consumption. However, considering this user-centric approach is problematic. Indeed, the works proposed in the literature all assume static usage patterns. In this, our approach is different, because it considers device-centric assumption yet with an adaptable model, constructed from real-world data, as explained in Section 4. This allows to build a robust baseline model of each device. In our vision, explained in Section 5, our approach can be extended by building user-centric models on top of such baselines. This is different than constructing either a device- or user-centric model.

Device-centric models Let us focus on the methodologies that use offline measurement in order to create a system-level power model [1]. [22] presents a methodology to isolate power consumption of the main hardware elements. In this work, a resistor is inserted in series between the battery and its connector on the phone, and a sampling board is used to measure the battery voltage. In [4], Carrol et al. present another work that relies on physical measurement. The goal of this work is to understand where and how energy is used, to provide a basis for managing energy consumption by physically measuring the power consumption of relevant, along with the overall power drain. They discovered that the majority of power consumption can be attributed to the GSM module, display, and graphics subsystem.

Another way to measure power on a device is to exploit interfaces already available on specific devices or OS, e.g. the *Advanced Configuration and Power Interface (ACPI)* (as proposed into this work). In [6] and [23] a self-modeling approach is presented, to build high-rate mobile system models without the need of external measurement systems. These works select data, as variables, that reflect the activity levels of each hardware component such as the hardware performance counters (HPCs) for processors, the downlink and uplink data rates for the wireless interfaces, and the brightness level for the display. These variables are used to perform a linear regression to generate the models.

In [24], Zhang et al. present a methodology that uses built-in battery voltage sensors and knowledge of battery discharge behavior to monitor power consumption while explicitly controlling the power management and activity states of individual components. They developed an Android application *PowerBooter* for online power estimation. One of the most interesting thing they discovered is that for components with significant power consumption, the states of other components become irrelevant: this means that if I want to know the consumption of two components it is simply possible to make a test on the first and on the second separately and sum the result. The power model in this work includes six components: CPU and LCD as well as GPS, Wi-Fi, audio, and cellular interfaces. The result indicates that the power model built with PowerBooter is accurate to within 4.1 % of measured values for 10-second intervals. The main problems about this work are, that it is necessary to have the specific discharge curve of the battery of that specific device to have a mobile phone with superuser access. For the first problem using the same look-up table for all batteries may be inaccurate. For the second problem the software cannot be used on every mobile device available on the market.

User-centric models Other works show how the power model of a mobile device is highly influenced by the user behavior, as widely discussed in [7]. In fact, in [9] the user perception and intention are considered a key factor for the creation of a device-specific power model. Also in [4], it is clearly shown how different kind of usage have different impact on the power consumption. The problem is the collection of subjective data; in fact, the power consumption and the user perception of the logging operation must be taken into account because it could results in a very power hungry task. Thus one of the main challenge is to collect as much information as possible, while keeping the direct interaction at minimum. Various technique have been used in literature to maximize the information provided by the users while keeping at minimum the interaction with them: a good example is the user feedback mechanism developed by Google on the Google Maps Navigation [20].

3 Power sensing component

Building accurate power models requires adequate amounts of data gathered from a diverse population of real users and devices. Such a logging system must be extensible (to support future version of the Android operating system) and efficient, with little impact on the battery life. Moreover, as the collected data is sent to a remote server, the communication also must be efficient. To this end, we used our MPower Android application as a power sensing tool, following a crowdsourcing-like approach.[2] We presented the vision and the design of MPower in [3, 19] and released the app on the Google Play Store in April, 2012. The

[2]https://play.google.com/store/apps/details?id=org.morphone.mpower

MPower user interface is shown in Fig. 1. We designed the application to be unobtrusive in terms of both operations required by the end user (i.e., no setup is required) and power needs (i.e., less than the 2 % of the overall battery power).

Although the main purpose of the application is to collect data for research purposes, as described in Section 3.1, we purposely introduced other important features that makes the application appealing and useful for the user. This is another important aspect for ensuring enough participants to crowdsourcing studies. An example of such useful features is the "smart suggestion" function exemplified in Fig. 1b, which provide tips to help users to save battery life. A trivial approach to this consists in turning off hardware subsystems (e.g., wifi, bluetooth or GPS) that have been unintentionally left on, despite unused. Typical scenarios include, for instance, a user commuting between home and work: Although both places probably offer Wi-Fi connectivity, keeping the antenna enabled while commuting turns out to be a waste of battery power. Similar mechanisms are described in details in the main work that present MPower [3].

As of June 2013, we have 278 active users and about 22.5 million records in our database. As summarized in Fig. 2, MPower collected data from a good variety of devices and operating system versions.

The reminder of this section describes what data MPower collects, how the logging works, and how we store data on on our servers.

3.1 Data collection and management

One of the most important aspects to consider when building a statistical power model (on mobile devices) is the data-gathering phase. On the one hand, the device battery life is influenced by several device components (e.g., number of running processes, battery charge level, voltage). On the other hand, collecting data from every such component is not a good practice: The more data is collected, the heavier the collection phase becomes in terms of battery consumption. As discussed in Section 3.2, an adequate tradeoff between the dimensionality and sampling frequency of the collected data must be found, in order to minimize the device resources affected, while ensuring that enough samples are collected, thus avoiding biased statistical models.

Upon the first installation, MPower records metadata such as the device brand and model, CPU architecture and kernel version. After that, MPower collects the following categories of data through the Android APIs. Specifically, MPower uses listeners and receivers to obtain new data only when a change occurs. This ensures that no machine cycles are wasted in this phase:

Battery. This includes the charging status, charge percentage, battery temperature, voltage and health. This information is essential to model the battery behavior.

Mobile network. The power consumption of a data connection is strongly influenced by two factors: the

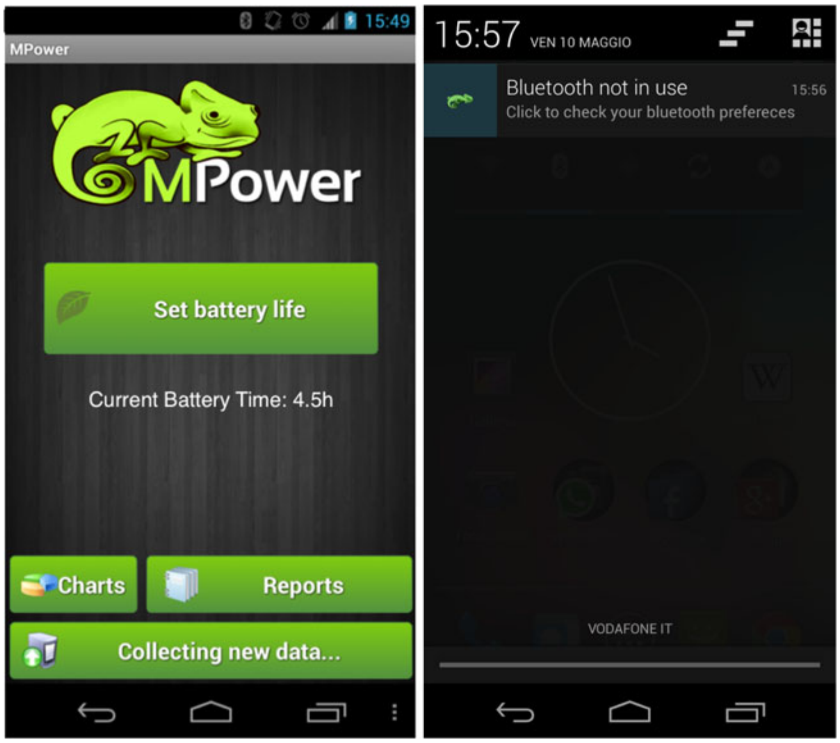
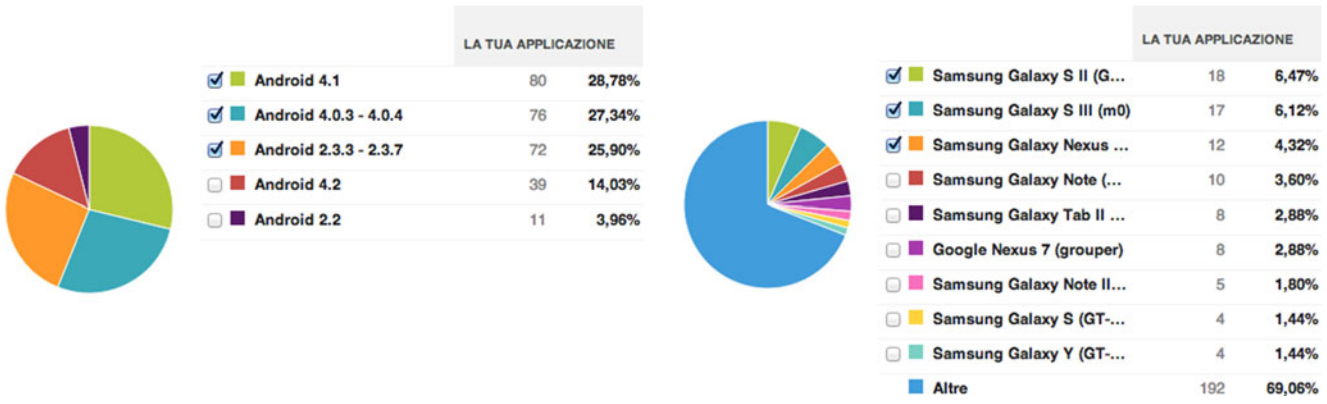


Fig. 1 MPower Android application user interface: **a** is a screenshot of the application main screen, and **b** is an example notification, which suggests to the user that the Bluetooth system should be disable because unused



		LA TUA APPLICAZIONE	
✓	🟩 Android 4.1	80	28,78%
✓	🟦 Android 4.0.3 - 4.0.4	76	27,34%
✓	🟧 Android 2.3.3 - 2.3.7	72	25,90%
☐	🟥 Android 4.2	39	14,03%
☐	🟪 Android 2.2	11	3,96%

		LA TUA APPLICAZIONE	
✓	🟩 Samsung Galaxy S II (G...	18	6,47%
✓	🟦 Samsung Galaxy S III (m0)	17	6,12%
✓	🟧 Samsung Galaxy Nexus ...	12	4,32%
☐	🟥 Samsung Galaxy Note (...	10	3,60%
☐	🟪 Samsung Galaxy Tab II ...	8	2,88%
☐	🟪 Google Nexus 7 (grouper)	8	2,88%
☐	🟥 Samsung Galaxy Note II...	5	1,80%
☐	🟨 Samsung Galaxy S (GT-...	4	1,44%
☐	🟩 Samsung Galaxy Y (GT-...	4	1,44%
	🟦 Altre	192	69,06%

Fig. 2 MPower statistics about devices that have currently installed MPower and are transmitted the collected data. Those statistics are taken from the official Google Play page

type of network used and the amount of data transferred. Therefore, MPower collects data about network type (e.g, GPRS, EDGE, UMTS), byes sent and received, interface condition (e.g., idle, connected, authenticating) and link speed. MPower collects also the call state and whether the airplane mode is active or not.

Wi-Fi network. Similarly to the previous category, we log the interface state, connection state, link speed and transmitted bytes.

Screen. The display is one of the most battery-demanding components. MPower logs whether or not the screen backlight is on, the brightness mode (automatic or manual) and value, as well as the screen refresh rate and size.

GPS data includes only whether or not the GPS is enabled and its current status (idle, connecting, transferring), which is a good approximation of the utilization. For privacy reasons we store no information about the user position.

Bluetooth. Similarly to GPS, MPower logs only whether or not the Bluetooth antenna is enabled, along with its internal status (idle, transmitting).

Audio interface data is also recorded. This includes the state of the microphone and speaker.

CPU power consumption is strongly influenced by its frequency. MPower collects data about the current frequency, along with minimum and maximum frequencies allowed, in conjunction with the governor policy in use. In case of multicore CPUs, MPower collects data about all the cores in use.

MPower ignores data regarding accelerometers, magnetometers and similar sensors, because these components are always active; therefore, it is not possible to act on them to optimize the battery life. In other words, we consider them as part of the battery's natural discharge curve.

3.2 Logging, storing and sending data

MPower logs and stores data by leveraging the *Andorid background services*. As mentioned above, the sampling interval is crucial. On the one hand, a small sampling period may seem to lead to more accurate models. On the other hand, two aspects must be considered: first, the device's dynamic can be slower than the chosen period; secondly, an excessively small period would in turn never trigger the sleep mode when necessary because of the high CPU load require by such a fast sampling frequency. This would ultimately result in high power leaks caused by MPower itself. According to our tests executed on different devices, and described in [3], revealed that a good tradeoff between data precision and the amount of power consumed by the collecting phase is around 10 s. In practice, we decreased the sampling rate until the MPower service permanently disappeared from the list of applications that contribute to more than the 2 % of the overall power consumption. This led us to finding a sampling period of 5 s, on the devices on our possession. Actually, we set a conservative value of 10 s.

The data listed in Section 3.1 is stored in four subsequent files per day (morning, afternoon, evening, night). Every 10 s, MPower appends a timestamped line to the current file. To guarantee privacy, each file is encrypted with AES, zipped, and sent to the server, which unzip it and store the contained records into a database for subsequent asynchronous processing. This happens automatically whenever Internet connectivity is available through a Wi-Fi network, or on demand (even via cellular data network). The symmetric key is exchanged during the authentication phase, which is performed through Google OAuth.

4 Power model generation

Our goal is to generate an accurate, device-centric power model. Previous work [4] either dealt with data collected in a controlled environment, thus creating non-adaptive, device-centric models, or with real data, thus creating user-centric power models [12–14]. MPower is profoundly different: It aims at creating a device-centric power model that still consider the dynamics typical of a mobile scenario, according to real-world data. To this end, we use the data collected as described in Section 3.

The remainder of this section describes the methodology we used to implement our power-model generation procedure on the server side, along with the results of the experimental evaluation.

4.1 Proposed methodology

The problem of power consumption estimation can be defined as follows: given an unknown discharge process $\mathcal{P}$ and an input-output sequence $Z = \{(x(t), y(t))\}_{t=1}^{N}$, with $x(t) \in \mathbb{R}^m$ and $y(t) \in \{0, \ldots, 100\}$, realization of $\mathcal{P}$, we want to find a function f able to predict the battery level $y(t)$, output of the process $\mathcal{P}$, given the past input $x(0, \ldots, t-1)$ and output $y(0, \ldots, t-1)$, or, more formally, to find:

$$\hat{y}(t) = f[y(0, \ldots, t-1), x(0, \ldots, t)]$$

where $y(0, \ldots, t-1)$ is the battery level in the time interval $\{0, \ldots, t-1\}$ and $x(0, \ldots, t)$ is the input history in the time interval $\{0, \ldots, t\}$. This is summarized in Fig. 3.

4.1.1 Model estimation approach

The prediction problem is addressed with multiple techniques in the literature, like in the time series prediction

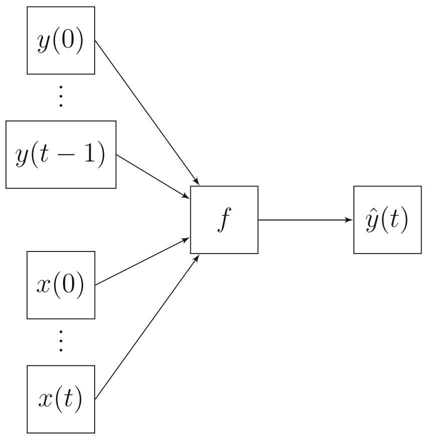


Fig. 3 Block diagram that describes the input-output relationships in our power model

[10] or the system identification fields [17]. In our case, we will perform an approximation of the original process $\mathcal{P}$ through an ARX, i.e., a linear autoregressive model with exogenous input (see [17] for details). In these kind of models, we define the one step ahead prediction function as a linear operator of the output past output y and inputs x:

$$\hat{y}(t) = \theta[y(t-k_a-n_a, \ldots, t-k_a-1), x(t-k_b-n_b, \ldots, t-k_b)]^T$$

where $\theta \in \mathbb{R}^{n_a+(n_b+1)m}$ is a parameter vector, k_a and k_b are the time delays and n_a and n_b the orders of the autoregressive and of the exogenous part, respectively. Non-linear behavior of the process $\mathcal{P}$ is addressed in Section 4.1.2. As summarized in Section 2.2, battery-life prediction is generally addressed in the literature by using linear regression models [18], i.e.:

$$\hat{y}(t) = \theta\, x(t)^T = \sum_{i=1}^{m} \theta_i x_i(t),$$

mainly because that this specific model requires a limited computation for prediction. In fact, once the model is trained, the battery level prediction requires a linear number of operation in the parameter vector dimension (e.g, $O(m)$), but it does not take temporal dependencies into account. In this work, we want to take into account also models that incorporate such dependencies. Therefore, for the model selection we will perform an exploration of the ARX models with different orders and time delays. The model selection phase will be conducted by basing on the Akaike Information Criterion (AIC), which balances the likelihood and the complexity of the model, which is defined as:

$$AIC = \log L - 2\ln(p),$$

where p is the number of free parameter of the model, i.e., $p = n_a + (m+1)n_b$, and L is the likelihood function of the estimated levels w.r.t the real one.

4.1.2 Nonlinearity management

It is quite clear that the linear choice in modeling may introduce a model bias in the estimation procedure, since we are not constraining the process $\mathcal{P}$ generating data to be linear. We addressed this problem using two different techniques. First, we discriminate between:

- the controllable variables $c = (c_1, \ldots, c_s)$;
- the uncontrollable variables $u = (u_1, \ldots, u_r)$

where $x(t) = [c(t), u(t)]$. This allow us to estimate a set of models $\bar{\theta}_{c^\star}$ by fixing the configuration $c^\star = (c_1^\star, \ldots, c_s^\star)$, where each one is approximating a subset of the device power discharge dynamics, leading to a better modeling of the process. Examples of configurations are listed in Table 1. Batches of data with continuous fixed configuration $Z_{c^\star}$ are then extracted: the estimation of each parameter will be conducted on these dataset. The splitting of the data with respect to the configurations would have no impact on a controlled experimental setting. However, in our approach, we must proceed with care, because it may result that some configurations are far less frequent than others, thus affecting the goodness of the model estimation. We address this as described in Section 4.1.4.

Second, we transform the uncontrollable variables into a feature space through a set of function $\phi_i(x_i, t) : \mathbb{R} \in \mathbb{R}^q$, which includes the possibility to exploit higher dimensional spaces to maintain a linear estimation model while keeping a low model bias. For instance, a possible feature space choice we considered was s.t. $\phi_i(x_i, t) = \sum_{t \in S} x_i(t)$, where S is the time interval in which a 1 % of the battery power was lost.

4.1.3 Model construction and feature selection

Despite the aforementioned problems deriving from use of linear models, their use will allow us to perform statistical comparison among learned power models. Furthermore, the knowledge of the parameter distribution allows us to apply a statistical feature-selection method. In fact, if we learn a sequence of power models parameters $\Theta = (\theta_1, \ldots, \theta_L)$ by using non overlapping batches of data $Z_1, \ldots, Z_L$ (of equal length N) for training, we are assured by the system identification theory [17] that they are asymptotically Gaussian distributed. If we consider a set of parameter vectors Θ, by computing:

$$\bar{\theta} = \sum_{l=1}^{L} \theta_l,$$

we will have a correct estimate of the optimal parameter in the approximating parameter space θ^o, w.r.t. the empirical risk (for details see [17]), and as a byproduct also an estimate S^2 of the real covariance matrix Σ^2 of the distribution. Then, we will be able to characterize the topology of the parameter space: this will allow us to perform statistical hypothesis tests for the significance of each parameter vector components [18]. In this phase we can exploit the known distribution of the parameters, to perform a feature selection on the exogenous variables. In fact, by analyzing the i-th component of the parameter vector $\bar{\theta}_i$, we have $\frac{\bar{\theta}_i - \theta_i^o}{s_{ii}} \sim \mathcal{N}(0, 1)$, we may test for:

$$H_0 : \theta_i = 0 \text{ vs. } H_1 : \theta_i \neq 0$$

where $\sigma_{ii} = \sqrt{\{S^2\}_{ii}}$. By basing on the critic region at significance level α, $R_\alpha = \{|t| > Z_{1-\alpha/2}\}$, where the test statistics is $t = \frac{\theta_i}{\sqrt{N} s_{ii}}$ and $Z_{\alpha/2}$ is the quantile of order $\alpha/2$, we can check for significance of each of the parameter we computed, and, consequently, the ability of each of the variables u we considered to predict the battery level y. Since we performed multiple test on different parameter vectors components, we will use the Bonferroni correction [18] to achieve an aggregate significance level.

As final recap, for the power model prediction we use internal configuration to select the specific *device state c* and each of them induce a model parameter $\bar{\theta}_c$ where the external conditions are used as *exogenous input*.

4.1.4 Device comparison

Relying again on the statistical properties of the parameter vectors $\bar{\theta}$, we are able to compare two devices on common configurations. Considering estimated parameter vectors coming from data gahtered from different devices (by fixing the same configuration) $\bar{\theta}^{(i)}$, $\bar{\theta}^{(i)}$, it is possible to design a statistical test:

$$H_0 : \bar{\theta}_i = \bar{\theta}_j \text{ vs. } H_1 : \bar{\theta}_i \neq \bar{\theta}_j$$

by basing on the critic region at significance level α,

$$\frac{L_i(L_j L_i - L_j - p + 1)}{(L_j + 1)(L_j - 1)p} \left(\bar{\theta}^{(i)} - \bar{\theta}^{(j)}\right)^T S^{-1} \left(\bar{\theta}^{(i)} - \bar{\theta}^{(j)}\right) \leq F_{p, L_i L_j - L_i - p + 1, \alpha}$$

where L_i and L_j are the number of parameter vectors used to estimate respectively $\bar{\theta}^{(i)}$ and $\bar{\theta}^{(j)}$ and $F_{p, L_i L_j - L_i - p + 1, \alpha}$ is the quantile of order $1 - \alpha$ of the Fisher's distribution (under the assumption that they share the covariance matrix, i.e., $\Sigma_i = \Sigma_j$). Either the test rejects H_0 or not we may infer important information:

- in the case the two devices are considerably different, we justify the estimation of the model for each particular device;
- if they are proven to be similar, we may built up a joint model, covering a set of configurations wider or equal to the two original one, which can be used for bot the devices.

This last consideration will allow our system to send a preliminary model to the device, in the case a similar one was already present in the database with a reliable model, and to refine it as soon as new data are recorded on the device.

4.2 Experimental evaluation

We considered data from 5 randomly chosen devices among those with at least 1 year of recorded data. We divided a dataset that represent a single device into its different configurations, as described in Section 4.1 (see also Table 1). For each configuration, we created a training set and a testing set.

Our preliminary analysis provided evidence that extracting models from $N = 100$ samples, corresponding to roughly 15 minutes of consecutive data, allows to be robust

Table 1 Configuration parameters: values and considered configurations

Parameter	Range	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
Airplane Mode	✓/✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Mobile Data State	✓/✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
WiFi	✓/✗	✗	✓	✓	✓	✗	✗	✗	✓	✓
Screen	✓/✗	✗	✗	✓	✓	✗	✓	✓	✗	✓
Bluetooth	✓/✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Brightness Level	0–100 %	0 %	0 %	0 %	66 %	0 %	0 %	66 %	0 %	0 %

with respect to noise (model modification is only 10 % even increasing data to $N = 400$), and to meet the Gaussian assumption.

Working with real-world data, rather than on data produced in a lab environments like previous work did, entails some difficulties. The most concrete problem is that we do not have a full discharge curve of a device in a fixed configuration: Likely, a user changes the phone configuration (see Table 1) during the day, and the battery discharge level y is influenced by such changes. Consequently, instead of using y, we compute d, which is the battery discharge level at time t, as:

$$d(t) = y(t) - y(t-1).$$

Therefore, our model becomes:

$$d(t) = \theta\, d(t-1)$$

where the order is selected according to the AIC index

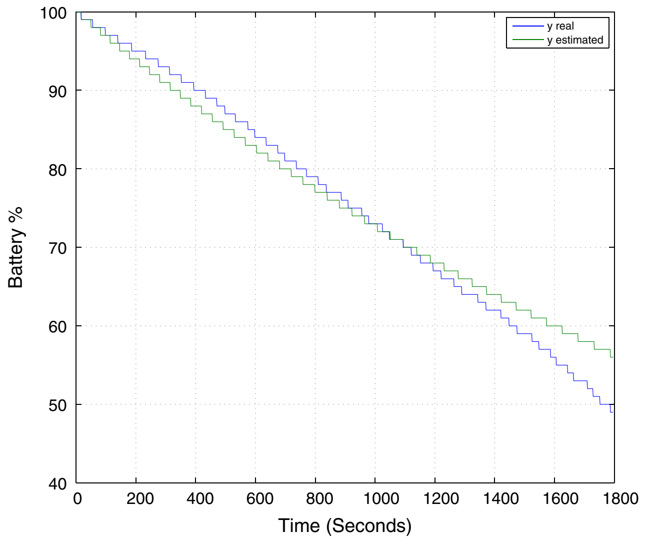


Fig. 4 Prediction of y, obtained from d as explained in Section 4.2. Note that, for easing the visual comparison between estimated and actual values, we rounded the prediction to an integer. According to our collected data, estimating the discharge rate as linear function is a viable option. Indeed, the mean error of our prediction is only 2 %

computed on our data as stated in Section 4.1. According to AIC, an AR(1) model is the best trade-off between precision and complexity.

Using this model, we estimated $\bar{\theta}$ as the average over different, equally-sized batches. By comparing our estimation with the measured data in the testing set, we obtained only a mean error of 2 % per hour. This experiment shows that the model is able to estimate the discharge dynamic with good precision, and, most importantly, leveraging a simple and "readable" model.

Figure 4 shows the prediction capabilities on y, obtained by reversing the aforementioned functional relationship between y and d. More precisely, we obtained this model from configuration c_1 (as of Table 1) on a temporal interval of 5 h.

In summary, we can conclude that, according to our collected data, estimating the discharge rate as linear function is a viable option.

Although these results are preliminary and obtained form a single device for validating our intuition, we believe that the generality and flexibility of our methodology easily allows for extending this work.

5 Conclusion

Despite the great complexity and technological, hardware and software advances in mobile devices development, smart applications are still bounded by their limited energy capacity. Therefore, battery is the most precious resource for mobile devices. As a consequence, there is an active research community focusing on how to design, develop and, most importantly, evaluate energy-aware software.

We believe that systems able to accurately characterize the power consumption on mobile devices are an important prerequisite in this research area. In this work we presented MPower, an adaptive power modeling system for Android, intended as fundamental component for energy-aware applications and research. The key difference of our approach consist in using a client-side power sensing component (i.e., the MPower Android application) to collect power-consumption data from real-world users and devices (i.e., 278 mobile devices, totaling about 22.5 millions of records collected in about 1 year of measurement).

Previous work either focused on data collected in a controlled environment, thus creating non-adaptive, device-centric models, or with real data, thus creating user-centric power models. Our approach allows us to build device-centric power models, which take into account the various hardware components, and how they contributed to the overall power consumption. In this, MPower is profoundly different from previous work. Indeed, we were able to create a device-centric power model that still consider the dynamics typical of a mobile scenario, according to real-world data.

In our vision, our approach will provide the users with tailored power models, toward better power-management strategies, or green security applications (for instance). In addition, our sensing and modeling technique is helpful to other researchers because it supports the empirical evaluation of energy-aware applications.

References

1. Anand M, Nightingale EB, Flinn J (2004) Ghosts in the machine: interfaces for better power management. In: Proceedings of the 2nd international conference on mobile systems, applications, and services, MobiSys '04. ACM, New York, p 23–35
2. Barnes R, Winterbottom J, Dawson M (2011) Internet geolocation and location-based services. Commun Mag IEEE 49(4):102–108
3. Bonetto A, Ferroni M, Matteo D, Nacci AA, Mazzucchelli M, Sciuto D, Santambrogio MD (2012) Mpower: towards an adaptive power management system for mobile devices. In: Proceedings of the 10th IEEE/IFIP international conference on embedded and ubiquitous computing (EUC), p 318–325
4. Carroll A, Heiser G (2010) An analysis of power consumption in a smartphone. In: Proceedings of the 2010 USENIX conference on USENIX annual technical conference, USENIXATC'10. USENIX Association, Berkeley
5. Caviglione L, Merlo A, Migliardi M (2011) What is green security? In: 2011 7th international conference on information assurance and security (IAS), p 366318–325371
6. Dong M, Zhong L (2011) Self-constructive high-rate system energy modeling for battery-powered mobile systems. In: Proceedings of the 9th international conference on Mobile systems, applications, and services, MobiSys '11. ACM, New York, p 335–348
7. Falaki H, Mahajan R, Kandula S, Lymberopoulos D, Govindan R, Estrin D (2010) Diversity in smartphone usage. In: Proceedings of the 8th international conference on Mobile systems, applications, and services, MobiSys '10, p 179–194
8. Ferroni M, Cazzola A, Matteo D, Trovo' F, Nacci AA, Sciuto D, Santambrogio MD (2013) Mpower: gain back your android battery life! In: Adjunct publication of the 2013 ACM conference on ubiquitous computing, UbiComp'13, 4 pages. Accepted to appear
9. Froehlich J, Chen MY, Consolvo S, Harrison B, Landay JA (2007) Myexperience: a system for in situ tracing and capturing of user feedback on mobile phones. In: Proceedings of the 5th international conference on Mobile systems, applications and services, MobiSys '07. ACM, New York, p 57–70
10. Hamilton JD Time series analysis, vol 2. Cambridge University Press, Cambridge
11. Hoffmann J, Neumann S, Holz T (2013) Mobile malware detection based on energy fingerprints—a dead end? In: Proceedings of the 16th international symposium on research in attacks, intrusions and defenses (RAID) (to appear)
12. Hong JWK, Kang J (2012) Method for predicting available time remaining on battery in mobile devices based on usage patterns. US Patent 8,204,552
13. Kang J-M, Seo SS, Hong JW-K (2011) Personalized battery lifetime prediction for mobile devices based on usage patterns. J Comput Sci Eng 5(4):338–345
14. Kang J-M, Seo SS, Hong JW-K (2011) Usage pattern analysis of smartphones. In: Network operations and management symposium (APNOMS), 2011 13th Asia-Pacific, p 1–8
15. Lane N, Miluzzo E, Peebles Lu H, Choudhury TD, Campbell A (2010) A survey of mobile phone sensing. Ad Hoc and Sensor Networks
16. Liu L, Yan G, Zhang X, Chen S (2009) Virusmeter: preventing your cellphone from spies. In: Proceedings of the 12th international symposium on recent advances in intrusion detection, RAID '09. Springer-Verlag, Berlin, Heidelberg, p 244–264
17. Ljung L (1987) System identification: theory for the user. Prentice-Hall information and system sciences series. Pearson Education Canada
18. Montgomery DC, Runger GC (2010) Applied statistics and probability for engineers. Wiley, New York
19. Nacci AA, Mazzucchelli M, Maggio M, Bonetto A, Sciuto D, Santambrogio MD (2013) Morphone.os: context-awareness in everyday life. In: Proceedings of EUROMICRO digital system design conference (DSD) (to appear)
20. Nakhimovsky Y, Miller AT, Dimopoulos T, Siliski M (2010) Behind the scenes of google maps navigation: enabling actionable user feedback at scale. In: CHI 2010 extended abstracts on human factors in computing systems. New York, NY, p 3763–3768
21. Pettey C, van der Meulen R (2012) Gartner says worldwide sales of mobile phones declined 3 percent in third quarter of 2012; smartphone sales increased 47 percent. http://www.gartner.com/newsroom/id/2237315
22. Rice AC, Hay S (2010) Decomposing power measurements for mobile devices. In: PerCom. IEEE Computer Society, p 70–78
23. Xiao Y, Bhaumik R, Yang Z, Siekkinen M, Savolainen P, Yla-Jaaski A (2010) A system-level model for runtime power estimation on mobile devices. In: Proceedings of the 2010 IEEE/ACM int'l conference on green computing and communications & int'l conference on cyber, physical and social computing, GREENCOM-CPSCOM '10. IEEE Computer Society, Washington, p 27–34
24. Zhang L, Tiwana B, Qian Z, Wang Z, Dick RP, Mao ZM, Yang L (2010) Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, CODES/ISSS '10. ACM, New York, p 105–114
25. Zhou Y, Xuxian J (2012) Dissecting Android malware: characterization and evolution. In: IEEE symposium on security and privacy, p 1–15