Catalog of Energy Patterns for Mobile Applications

Vinicius Bezerra
Matheus Casa Nova

Energy Patterns for Mobile Apps

Web

Online Catalog with 22 patterns to improve the energy efficiency of iOS and Android apps.

Awesome Mobile App Energy Papers

Web

A curated list of awesome papers that study energy efficiency for mobile applications.

Download Curriculum Vitae

PDF

Updated on February, 2019.

Publications

Luis Cruz and Rui Abreu (2019). **On the Energy Footprint of Mobile Testing Frameworks**. *IEEE Transactions on Software Engineering*.

Luis Cruz and Rui Abreu and John Grundy and Li Li and Xin Xia (2019). **Do Energy-oriented Changes Hinder Maintainability?**. In *ICSME*. (Slides)

Luis Cruz, Rui Abreu (2019). **EMaaS: Energy Measurements as a Service for Mobile Applications**. In *41st International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*.

Luis Cruz, Rui Abreu and David Lo (2019). **To the Attention of Mobile Software Developers: Guess What, Test your App!**. In *Empirical Software Engineering*.

Luis Cruz and Rui Abreu (2019). **Catalog of Energy Patterns for Mobile Applications**. In *Empirical Software Engineering*.

Luis Cruz and Rui Abreu (2018). **Using Automatic Refactoring to Improve Energy Efficiency of Android Apps**. In *CIbSE XXI Ibero-American Conference on Software Engineering*. Best Paper Award

Luis Cruz and Rui Abreu (2017). **Performance-based Guidelines for Energy Efficient Mobile Applications**. In *IEEE/ACM International Conference on Mobile Software Engineering and Systems, MobileSoft 2017*. (Slides)

Luis Cruz and Rui Abreu and Jean-Noël Rouvignac (2017). **Leafactor: Improving Energy Efficiency of Android Apps via Automatic Refactoring**. In *IEEE/ACM International Conference on Mobile Software Engineering and Systems, MobileSoft 2017*. (Slides)

Luis Cruz and Jonathan Rubin and Rui Abreu and Shane Ahern and Hoda Eldardiry and Daniel G. Bobrow (2015). **A wearable and mobile intervention delivery system for individuals with panic disorder**. In *Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia* (pp. 175--182).

Strecht, Pedro and Cruz, Luis and Soares, Carlos and Mendes-Moreira, Joao and Abreu, Rui (2015). **A Comparative Study of Regression and Classification Algorithms for Modelling Students' Academic Performance**. *Educational Data Mining 2015*.

Cruz, Luis and Reis, Luis Paulo and Rei, Luis (2011). **Generic optimization of humanoid robots' behaviours**. In *15th Portuguese Conference on Artificial Intelligence, EPIA* (pp. 385--397).

Cruz, Luis and Reis, Luis Paulo and Lau, Nuno and Sousa, Armando (2012). **Optimization approach for the development of humanoid robots' behaviors**. *Advances in Artificial Intelligence--IBERAMIA 2012* (pp. 491--500). Springer Berlin Heidelberg.

Extras

Slides of my lecture on **Data Cleaning** at DELix 2018 SlideShare

Resumo

RQ1: Which design patterns do mobile app developers adopt to improve energy efficiency?

RQ2: How different are mobile app practices addressing energy efficiency across different platforms?

Catalogou Padrões de projetos voltados para eficiência energética

22

Coletou Apps Open-Source relacionados com o tema

- **1027 Android**
- **756 iOS**

Trabalhos Relacionados

- **Trabalhos voltados para otimizações em baixo nível, drivers e kernels em C,C++/Java**

- **Trabalhos de eficiência energética focados em apps de navegação**

- **Posts do stackoverflow**

Contribuições

- Catálogo de Padrões de projeto em linguagens alto-nível como Swift, Kotlin e Java

Metodologia

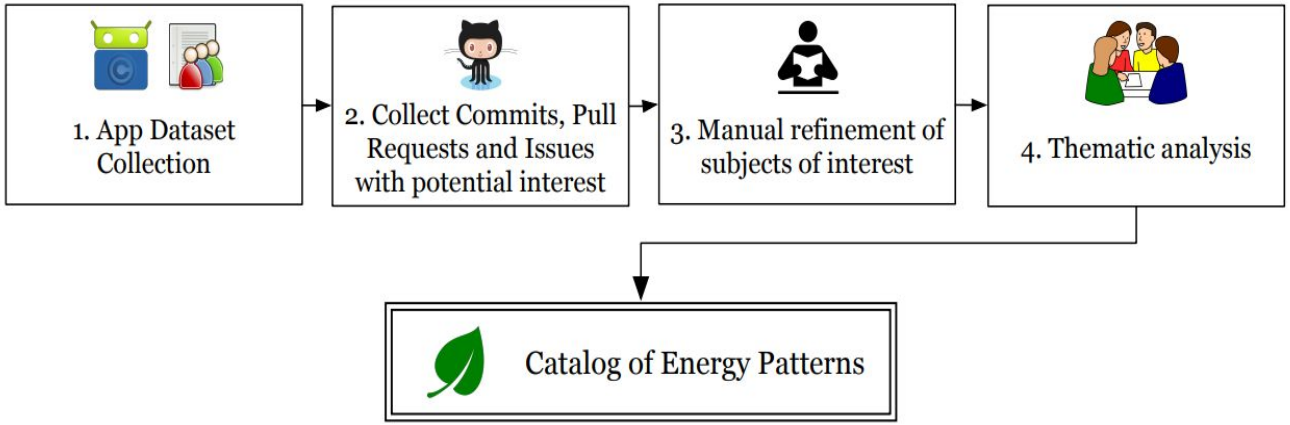


Fig. 1: Methodology used to extract energy patterns from mobile apps.

1. App Dataset Collection

- Coletou dataset de apps open-source
- 1027+756=1783 apps

1. App Dataset Collection

- *F-droid* *(https://f-droid.org)*
- *Community-curated collections of Android open source apps*
  (https://github.com/Mybridge/amazing-android-apps)

1. App Dataset Collection

- ***Collaborative List of Open-Source iOS Apps*** ***(https://github.com/dkhamsing/open-source-ios-apps)***

2. Collect Commits, Pull Requests and Issues with potential interest

```
.*(energy|battery|power).*
```

- *1783 apps → 6028 matches*

3. Manual refinement of subjects of interest

- *"Adding a link to the apps device settings in iOS Settings.app would be great for **power** users."* (False positive found in the app *WordPress* for iOS[8]).
- *"(...) recently a lot of issues that the core team does not have the **energy** to implement themselves have been closed."* (False positive found in the app *Minetest* for Android[9]).
- *"One thing is really important on mobile devices, and that is **power** consumption"* (True positive found in the app *ChatSecure* for iOS[10]).

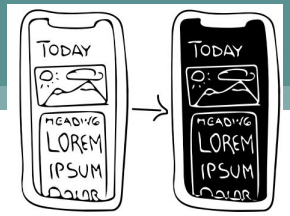
- **6028 → 1563 matches**
  - **332 commits**
  - **1089 issues**
  - **142 PRs**

4. Thematic analysis

- ***Metodologia sugerida em papers baseada em 4 passos:***
  - ***Familiarization with data***
  - ***Generating initial labels***
  - ***Reviewing themes***
  - ***Defining and naming themes***

Table 2: Energy patterns' occurrences and related work.

Pattern	Android	iOS	Related Work	Grey Literature
Dark UI Colors	28	2	(Agolli et al., 2017; Linares-Vásquez et al., 2017; Li et al., 2014, 2015)	-
Dynamic Retry Delay	10	2	-	-
Avoid Extraneous Work	32	9	-	[1]
Race-to-idle	27	5	(Liu et al., 2016; Banerjee and Roychoudhury, 2016; Cruz and Abreu, 2017; Pathak et al., 2012b)	-
Open Only When Necessary	4	3	(Banerjee and Roychoudhury, 2016; Reimann et al., 2014)	-
Push over Poll	13	3	-	[2, 3]
Power Save Mode	24	5	-	[4]
Power Awareness	35	6	(Bao et al., 2016)	[4]
Reduce Size	3	0	(Boonkrong and Dinh, 2015)	[5]
WiFi over Cellular	13	2	(Metri et al., 2012)	[6, 7, 8]
Suppress Logs	7	1	(Chowdhury et al., 2018b)	-
Batch Operations	17	1	(Li and Halfond, 2014; Corral et al., 2015; Cai et al., 2015)	[9, 10, 11]
Cache	14	4	(Gottschalk et al., 2014)	[10]
Decrease Rate	27	10	-	-
User Knows Best	33	11	-	-
Inform Users	6	4	-	-
Enough Resolution	10	7	-	-
Sensor Fusion	12	3	(Shafer and Chang, 2010)	[12]
Kill Abnormal Tasks	11	1	-	-
No Screen Interaction	8	2	-	-
Avoid Extraneous Graphics and Animations	11	8	(Kim et al., 2016)	[1]
Manual Sync, On Demand	4	5	-	-



4.1 Dark UI Colors

Provide a dark UI color theme to save battery on devices with AMOLED[13] screens (Agolli et al., 2017; Linares-Vásquez et al., 2017; Li et al., 2014, 2015).

Context: Screen is one of the major sources of power consumption in mobile devices. Apps that require heavy usage of screen (e.g., reading apps) can have a substantial negative impact on battery life.

Solution: Provide a UI with dark background colors, as illustrated in Figure 4. This is particularly beneficial for mobile devices with AMOLED screens, which are more energy efficient when displaying dark colors. In some cases, it might be reasonable to allow users to choose between a light and a dark theme. The

4.2 Dynamic Retry Delay

Whenever an attempt to access a resource fails, increase the time interval before retrying to access the same resource.

Context: Mobile apps that need to collect or send data from/to other resources (e.g., update information from a server). Commonly, when the resource is unavailable, the app will unnecessarily try to connect the resource for a number of times, leading to unnecessary power consumption.

Solution: Increase retry interval after each failed connection. It can be either a linear or exponential growth. Update interval can be reset upon a successful connection or a given change in the context (e.g., network status).

Example: Consider a mobile app that provides a news feed and the app is not able to reach the server to collect updates. Instead of continuously polling the server until the server is available, use the Fibonacci series[14] to increase the time between attempts.

4.3 Avoid Extraneous Work

Avoid performing tasks that are either not visible, do not have a direct impact on the user experience to the user or quickly become obsolete. This has been documented in the iOS online documentation[15].

Context: Mobile apps have to perform a number of tasks simultaneously. There are cases in which the result of those tasks is not visible (e.g., the UI is presenting other pieces of information), or the result is not necessarily relevant to the user. This is particularly critical when apps go to the background. Since the data quickly becomes obsolete, the phone is using resources unnecessarily.

Solution: Select a concise set of data that should be presented to the user and enable/disable update and processing tasks depending on their effect on the data that is visible or valuable to the user.

Example: Consider a time series plot that displays real-time data. The plot needs to be constantly updated with the incoming stream of data — however, if the user scrolls up/down in the UI view making the plot hidden, the app should cease drawing operations related with the plot.

4.4 Race-to-idle

Release resources or services as soon as possible (such as wake locks, screen) (Liu et al., 2016; Banerjee and Roychoudhury, 2016; Cruz and Abreu, 2017; Pathak et al., 2012b).

Context: Mobile apps use a number of resources that can be manually closed after use. While active, these resources are ready to respond to requests from the app and require extra power consumption.

Solution: Make sure resources are inactive when they are not necessary by manually closing them.

Example: Implement handlers for events that are fired when the app goes to background, and release wake locks accordingly.

4.5 Open Only When Necessary

Open/start resources/services only when they are strictly necessary.

Context: Some resources require to be opened before use. It might be tempting to open the necessary resources at the beginning of some task (e.g., upon the creation of an activity). However, resources will be actively waiting for requests, and consequently consuming energy.

Solution: Open resources only when necessary. This also avoids activating resources that will never be used (Banerjee and Roychoudhury, 2016).

Example: In a mobile app for video calls, only start capturing video at the moment that it will be displayed to the user[16] .

4.6 Push over Poll

Use push notifications to receive updates from resources, instead of actively querying resources (i.e., polling).

Context: Mobile apps need to get updates from resources (e.g., from a server). One way of checking for updates is by periodically query those resources. However, this will lead to several requests that will return no update, leading to unnecessary energy consumption.

Solution: Use push notifications to get updates from resources. Note – this is a big challenge amongst FOSS apps since there is no good open source alternative for Firebase Cloud Messaging (former Google Cloud Messaging).

Example: In a messaging app, instead of actively check for new messages, the app can subscribe push notifications.

4.7 Power Save Mode

Provide an energy efficient mode in which user experience can drop for the sake of better energy usage.

Context: Whenever the device battery is running low, users want to avoid losing connectivity before they reach a power charging station. If the device shuts down, users might miss important calls or will not be able to do an important task. Still, apps might be running unimportant tasks that will reduce battery life in this critical context.

Solution: The app provides a power save mode in which it uses fewer resources while providing the minimum functionality that is indispensable to the user. It can be activated manually or upon some power events (e.g., when battery reaches a given level). User experience can drop for the sake of energy efficiency. Note, this is enforced in iOS for some use cases if the apps use the BackgroundSync APIs.

Example: Deactivate features, reduce update intervals, or deactivate animated effects in the UI.

4.8 Power Awareness

Have a different behavior when the device is connected/disconnected to a power station or has different battery levels.

Context: There are some features that are not strictly necessary to users although they improve user experience (e.g., UI animations). Moreover, there are operations that do not have high priority and do not need to execute immediately (e.g., backup data in the cloud).

Solution: Enable or disable tasks or features according to power status. Even when the device is connected to power, the battery might still be running low, it might be advisable to wait until a pre-defined battery level is reached (or the power save mode is deactivated).

Example: Delay intensive operations such as cloud syncing or image processing until the device is connected to a charger.

4.9 Reduce Size

When transmitting data, reduce its size as much as possible.

Context: Data transmission is a common operation in mobile apps. However, such operations are energy greedy and the time of transmission should be reduced as much as possible.

Solution: Exchange only what is strictly necessary, avoiding sending unnecessary data. Use data compression when possible.

Example: When performing HTTP requests, use gzip content encoding to compress data.

4.10 WiFi over Cellular

Delay or disable heavy data connections until the device is connected to a WiFi network.

Context: Data needs to be synchronized with a server but it is not urgent and can be postponed.

Solution: Data connections using cellular networks are usually more battery intensive than connections using WiFi (Metri et al., 2012). Low priority operations that require a data connection to exchange considerable amounts of data should be delayed until a WiFi connection is available.

Example: Consider a mobile app to organize photos that allows users to backup their photos in a cloud server. Use an API to check the availability of a WiFi connection and postpone cloud synchronizing in case it cannot be reached.

4.11 Suppress Logs

Avoid using intensive logging. Previous work has found that logging activity at rates above one message per second significantly reduces energy efficiency (Chowdhury et al., 2018b).

Context: Developers resort to logging in their mobile apps to ensure their correct behavior and simplify bug reporting. However, logging operations create overhead on energy consumption without creating value to the end user.

Solution: Avoid using intensive logging, keeping rates below one message per second.

Example: Disable logging when processing real-time data. If necessary enable only during debugging executions.

4.12 Batch Operations

Context: Executing operations separately leads to extraneous tail energy consumptions (Li and Halfond, 2014; Corral et al., 2015; Cai et al., 2015). As illustrated in Figure 6, executing a task often induces tail energy consumptions related with starting and stopping resources (e.g., starting a cellular connection).

Solution: Bundle multiple operations in a single one. By combining multiple tasks, tail energy consumptions can be optimized. Although background tasks can be expensive, very often they have flexible time constraints. I.e., a given task needs to be eventually executed, but it does not need to be executed in a specific time.

Example: Use Job Scheduling APIs (e.g., 'android.app.job.JobScheduler', 'Firebase JobDispatcher') that manage multiple background tasks occurring in a device. These APIs will guarantee that the device will exit sleep mode only when there is a reasonable amount of work to do or when a given task is urgent. It combines several multiple tasks to prevent the device from constantly exiting sleep mode (or doze mode). Other examples: execute low priority tasks only if another task is using the same required resources; try to collect location data when other apps are collecting it as well.

4.13 Cache

Avoid performing unnecessary operations by using cache mechanisms.

Context: Typically mobile apps present data to users that is collect from a remote server. However, it may happen that the same data is being collected from the server multiple times.

Solution: Implement caching mechanisms to temporarily store data from a server (Gottschalk et al., 2014). In addition, verify whether there is an update before downloading all data.

Example: Considering a social network app that shows other users' profiles. Instead of downloading basic information and profile pictures every time a given profile is opened, the app can use data that was locally stored from earlier visits.

4.14 Decrease Rate

Increase time between syncs/sensor reads as much as possible.

Context: Mobile apps have to periodically perform operations. If the time between two executions is small, the app will be executing operations more often. In some cases, even if operations are executed more often, it will not affect users' perception.

Solution: Increase the delay between operations to find the minimal interval that does not compromise user experience. This delay can be manually tuned by developers or defined by users. More sophisticated solutions can also use context (e.g., time of the day, history data, etc.) to infer the optimal update rate.

Example: Consider a news app that collects news from different sources, each one having its own thread. Some news sources might have new content only once a week, while others might be updated every hour. Instead of updating all threads at the same rate, use data from previous updates to infer the optimal update rate of these threads. Connect to the news source only if new updates are expected.

4.15 User Knows Best

Allow users to enable/disable certain features in order to save energy.

Context: Energy efficiency solutions often provide a tradeoff between features and power consumption. However, this tradeoff is different for different users — some users might be okay with fewer features but better energy efficiency, and vice versa.

Solution: Allow users to customize their preferences regarding energy critical features. Since this might be more intuitive for power users, mobile apps should provide optimal preferences by default for regular users.

Example: Consider a mail client for POP3 accounts as an example. In some cases, users are not expecting any urgent message and are okay with checking for new mail in no less than 10 minutes for the sake of energy efficiency. On the other hand, there are cases in which users are waiting for urgent messages and would like to check for messages every two minutes. Since there is no automatic mechanism to infer the optimal update interval, the best option is to allow users to define it.

4.16 Inform Users

Let the user know if the app is doing any battery intensive operation.

Context: There are specific use cases in mobile apps that can be energy greedy. On the other hand, some features might be dropping user experience in order to improve energy efficiency. If users do not know what to expect from the mobile app, they might think it is not behaving correctly.

Solution: Let users know about battery intensive operations or energy management features. Properly flag this information in the user interface (e.g., alerts).

Example: Alert users when a power saving mode is active, or alert when a battery intensive operation is about to be executed.

4.17 Enough Resolution

Collect or provide high accuracy data only when strictly necessary.

Context: When collecting or displaying data, it is tempting to use high resolutions. The problem of using data with high resolution is that its collection and manipulation require more resources (e.g., memory, processing capacity, etc.). As a consequence, energy consumption increases unnecessarily.

Solution: For every use case, find the optimal resolution value that is required to provide the intended user experience.

Example: Consider a running app that is able to record running sessions. While the user is running, the app presents the current overall distance in real-time. While calculating the most accurate value of the total distance would provide the correct information, it would require precise real-time processing of GPS or accelerometer sensors, which can be energy greedy. Instead, a lightweight method could be used to estimate this information with lower but reasonable accuracy. At the end of the session, the accurate results would still be processed, but without real-time constraints.

4.18 Sensor Fusion

Use data from low power sensors to infer whether new data needs to be collected from high power sensors

Context: Mobile apps provide features that require reading data or executing operations in different sensors or components. Such operations can be energy greedy, causing high power consumption. Thus, they should be called as fewer times as possible.

Solution: Use complementary data from low power sensors to assure whether a given energy-greedy operation needs to be executed.

Example: Use the accelerometer to infer whether the user has changed location. In the case that the user is in the same location, data from GPS does not need to be updated.

4.19 Kill Abnormal Tasks

Provide means of interrupting energy greedy operations (e.g., using timeouts, or users input).

Context: Mobile apps might feature operations that can be unexpectedly energy greedy (e.g., taking a long time to execute).

Solution: Provide a reasonable timeout for energy greedy tasks or wake locks. Alternatively, provide an intuitive way of interrupting those tasks.

Example: In a mobile app that features an alarm clock, set a reasonable timeout for the duration of the alarm. In case the user is not able to turn it off it will not drain the battery.

4.20 No Screen Interaction

Whenever possible allow interaction without using the display.

Context: There are apps that require a continuous usage of the screen. However, there are use cases in which the screen can be replaced by less power intensive alternatives.

Solution: Allow users to interact with the app using alternative interfaces (e.g., audio).

Example: In a navigation app, there are use cases in which users might be only using audio instructions and do not need the screen to be on all the time. This pattern is commonly adopted by audio players that use the earphone buttons to play/pause or skip songs.

4.21 Avoid Extraneous Graphics and Animations

Graphics and animations are really important to improve the user experience. However, they can also be battery intensive — use them with moderation (Kim et al., 2016). This is also a recommendation in the official documentation for iOS developers[17]

Context: Mobile apps often feature impressive graphics and animations. However, they need to be properly tuned in order to prevent battery drain of users' devices. This is particularly critical in e-paper devices.

Solution: Study the importance of graphics and animations to the user experience. The improvement in user experience may not be sufficient to cover the overhead on energy consumption. Avoid using graphics animations or high-quality graphics. Resort to low frame rates for animations when possible.

Example: For example, a high frame rate may make sense during game play, but a lower frame rate may be sufficient for a menu screen. Use a high frame rate only when the user experience calls for it.

4.22 Manual Sync, On Demand

Perform tasks exclusively when requested by the user.

Context: Some tasks can be energy intensive, but not strictly necessary for some use cases of the app.

Solution: Provide a mechanism in the UI (e.g., button) that allows users to trigger energy intensive tasks.

Example: In a beacon monitoring app, there are occasions in which the user does not need to keep track of her/his beacons. Allow the user to start and stop monitoring manually.

Conclusões

Apps com padrões de eficiência energética

- **25% Android**
- **10% iOS**

- *Avoid Extraneous Work*
- *Decrease Rate*
- *User Knows Best*
- *Avoid Extraneous Graphics and Animations*

- *Power Awareness*
- *Dark UI Colors*
- *User Knows Best*
- *Race-to-Idle*

- *Frequência similar em ambas*
  - *Inform Users*
  - *Enough Resolution*
  - *Avoid Extraneous Graphics and Animations*