Power Aware Computing

Mechanisms for Power Draw Analysis on Mobile Devices

>_ DEV v1.3-RC1

Ernesto Martínez García [1]
me@ecomaikgolf.com

Marcell Haritopoulos [2]
marcell.haritopoulos@student.tugraz.at

Graz University of Technology

Power Aware Computing LU SS23

23rd of November 2023

SLIDES & REPORT

ls.ecomaikgolf.com/slides/power-aware-comp

Motivation

Why Mobile Energy Consumption

- Mobile Devices took big part of the computing share in the last decade
- Big part of the modern economy & society is based on them
- Our smartphones runs without a power plug. The best battery wins the market

Objectives

- Show how devices we use daily manage power consumption
- Overview the paper as a baseline, but base the research on current techniques
- Android & Mobile Devices hardware update almost each year. Check curent state

Table of Contents

Introduction 📘

History of Batteries

- Mobile Devices heavily depend on batteries to run
- In the case of smarphones, they condition the quality of the product
  - More battery means more weight & size
  - More battery means more `CPU/GPU/TPU` power
  - More battery means more charging times
  - More battery means more usage time
  - And the respective inverses
- Smartphone battery is the feature that increased the least between 2010 and 2019
- In some cases, vendors can’t increase the size the battery due to limitations
- As a result, Consumption Optimization plays a leading role in mobile devices

User Endpoint Importance

Who’s using your product?

- A contract bussiness partner
- A skilled user
- A non skilled user

Can we optimize power the same way for all of our products?

- Bussiness partner probably runs our code as intended by contract
- Skilled users might follow our guidelines for optimization
- Non skilled user might code infinite `while(1)` loops

See how there is a need for consumption control in the device

This task is commonly done by at Operating System level

Model-based
Energy Profiling

Introduction

❓ What's "Model-based Energy Profiling" about?

We leverage the following mechanisms/components:

Power Measure	Power Model	Power Estimation	Power Profiler

To profile the energy consumption of:

System	Subcomponents	Applications

If we think about the challenge it proposes, it's tricky

We have a system-wide power measurement

We could have per-component measurement, but components are reused for tasks

How we identify which consumption comes from where?

Model-based Energy Profiling Definitions

Power Measurement

Obtaining power (or current) consumption values directly from hardware.

Power Model

Mathematical model of power draw with variables that quantify the impact of usage

Power Estimation

Power draw consumption of a specific subsystem based on the Power Model.

Power Profiler

Leverage previous techniques to estimate power usage at certain abstraction layers

Constructing a Model-based Energy Profiler

The development of it is divided in four phases

Method Select.	Variable Selection	Model Train.	Profile Evaluation

Some of them are done in Laboratories some on a Computer

A general overview of the process would be:

1. An expert chooses a Method
2. From the Method we get Models
3. An expert selects Variables for the Model
4. With the Parameterized Model, we train it with device "logs"
5. The Fitted Model output is evaluated

System-Level Power Consumption Retrieval

External instrumentation

- Idea: Measure overall power consumption using external tools
- Multitude of options available:

Power Monitor

Connects to the device's battery connector and directly powers it, while allowing the monitor to measure the draw.

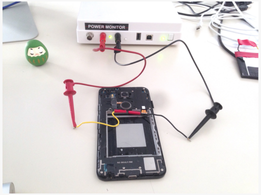


Figure: Example usage of Monsoon Power Monitor[1]

[1] Source: `https://tqrg.github.io/physalia/monsoon_tutorial`

External instrumentation

- Idea: Measure overall power consumption using external tools
- Multitude of options available:

Voltage Meter

Place a resistor in series to the power source, use Ohm's Law to deduce current.
Can operate on either the device's battery or external supply (but external supply is preferred as battery voltage changes with state of charge)

Figure: NEAT power meter installed in a phone[1, p. 8]

External instrumentation

💡 Idea: Measure overall power consumption using external tools

⚠ Mostly feasible only in laboratory environment:

- All of these methods at least require to open the phone
  → Potentially destructive on some phones (e.g. adhesive and IP sealing)
- Mostly not portable, especially when not using the device's battery

Self-Metering

Idea: Include capabilities in the phone to deduce power consumption without tools

Open circuit voltage (OCV)

The voltage of the battery with no load attached.

State of Charge (SOC)

The remaining charge capacity the battery holds, in percent.

By measuring change in SOC over an interval, we can deduce the average current draw using the OCV.

But: Batteries are not ideal
→ Battery models necessary

Self-Metering - Battery models

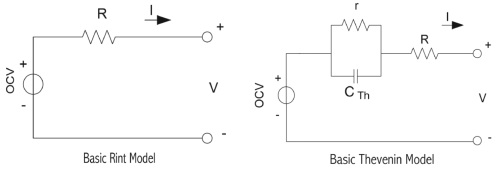


Figure: Battery models[3, p. 39:6]

Terminal voltage (V_t)

The voltage measurable across the battery's terminals.

⚠ Due to internal components, the terminal voltage drops depending on the current flow

⚠ As a consequence, we are unable to measure true OCV (but come closet with low power draw)

Self-Metering - Battery models

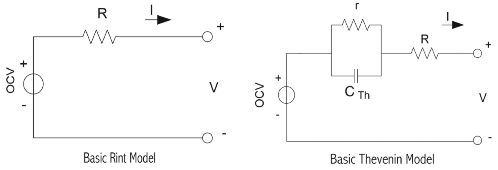


Figure: Battery models[3, p. 39:6]

Rint Model

Single ohmic resistance in series.

$$V_t = V_{OCV} - I * R$$

Thevenin Model

Additional capacitor models transient response of charging and discharging.

$$V_t = V_{OCV} - V_c - I * R$$

SOC estimation

SOC can usually not be measured - it needs to be estimated

As it is estimated, estimation errors propagate to energy profiles

There are two methods available:

Voltage-based estimation

Use strictly decreasing discharge curve to map OCV to SOC.

Mapping is fragile and depends on the batteries properties (age, model, ...)
→ Personalized discharge curve must be updated regularly

Battery voltage depends on a lot of factors: Age, temperature, (dynamic) load, ...
→ Significant estimation error

SOC estimation

SOC can usually not be measured - it needs to be estimated

As it is estimated, estimation errors propagate to energy profiles

There are two methods available:

Coulomb counting

Accumulate drawn current over time by directly sensing current:

$$SOC = SOC_{init} - \int \frac{I_{bat}}{C_{useable}} \, dt$$

Usable battery capacity $C_{useable}$ must be estimated and depends on various factors: Age, temperature, charge cycles, ...

Offset current accumulation error due to ADC must be compensated (e.g. long idle times)

Power Modeling Methodology

Introduction

- Power Models measure power consumption based on different inputs

  Models are usually pre-trained with inputs & logs measurements

  This yields a mathematical model that for a certain input, can estimate power

- Power Models obviously have an error percentage

  Error can be obtained by comparing results with hardware-instrumented devices

- Depending on input type, different classifications arise

- Depending on available information, different classifications arise

Input-type Classification

Utilization Based Models

Correlates power draw with measured resource usage.

An example of this mechanism is CPU’s Hardware Performance Counters.

Event-Based Models

Captures the power draw based on events (syscalls, state changes, …).

Useful for HW components with nonlinear power draw.

Handles tail energy better: Some devices delay entering sleep when no longer in use.

</> Code Analysis Based Models

Estimate power by inputing the source code via static analysis.

Information-Available Classification

Whitebox Modeling

We can capture consumption behaviour with Finite State Machines (FSM).

FSM's would describe the consumption in each state and the cost of transitioning them

We require knowledge of the states and the triggers for transitioning them

Usecase: Modeling the wireless subcomponent, as different states are important

Blackbox Modeling

We can't capture high granularity specific consumption behaviour.

We rely on models like linear regression with fitting to get an approximation

Assumption of linearity comes with limitations.

Usecase: Modeling the screen brightness subcomponent.

Energy Profilers

Categories of Energy Profilers

- Energy profilers can be divided into three categories:
  - On-device profilers with on-device model construction
  - On-device profilers with off-device model construction
  - Off-device profilers (with off-device model construction)
- No research about off-device profilers with on-device model construction is known

  Copying model for offline processing cubersome; processing can be done on device

On-device profilers with on-device model construction

e.g.: Nokia NEP (Symbian), Qualcomm Trepn, PowerBooter, Sesame, ...

- These profilers rely on battery state updates through self-metering
- Only one profiler needed some external calibration (but no external measurments), all others are independent from external tooling.
- Most profilers use linear regression models, but some profilers automatically generate models
- Various unique approaches (voltage vs. current, special HW support, ...)

- Qualcomm's Trepn is most accurate (close to Monsoon Power Monitor)!
  - Snapdr. SoCs has hardware instrum. using sense resistors and & fuel gauge in PMIC
  - Able to record fine-grained subcomponent energy consumption (CPU, GPU, ...)

On-device profilers with off-device model construction

e.g.: Android Power Profiler, PowerTutor, PowerProf

- Depend on vendor-provided offline calibration and power measurement phases
- Integral part of mobile operating system
- Profiles vary on the supported HW components and states, affecting accuracy

---

- Android's Power Profiler is discussed later
- PowerProf is able to learn model unsupervised with genetic algorithms

Off-device profilers

e.g.: PowerScope, JouleWatcher, Eprof, ...

Profile app's resource utilization (time and/or syscalls), performs code analysis (on-device or emulated)

Generate model by mapping activities to energy consumption based on utilization or FSMs.

Usually supports accurate and fine energy consumption characterization of app, subsystem and device (suitable for debugging apps)

Generate model by mapping activities to energy consumption

Energy Diagnosis

Energy Bugs

Energy Bug (eBug)

"[A]n error in the system, either application, OS, hardware, firmware or external that causes an unexpected amount of high energy consumption by the system as a whole"[5, p. 1]

- eBugs are not traditional bugs: Comptation and stability unaffected
- Hard to detect and pinpoint due to diversity of causes:

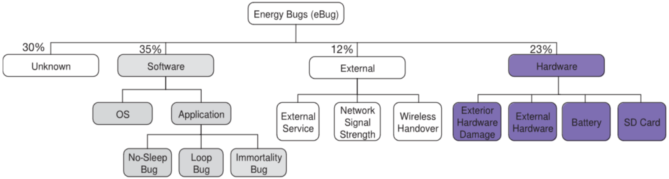


Figure: Categorization of eBugs [3, p. 39:25]

Energy Bugs

Energy Bug (eBug)

"[A]n error in the system, either application, OS, hardware, firmware or external that causes an unexpected amount of high energy consumption by the system as a whole"[5, p. 1]

Energy diagnosis engine can give insights in eBugs

Depending on the tool, answers (some of) these questions:

- What is the normal power usage of the program? What is abnormal?
- Is power optimization beneficial?
- Is higher power draw caused by the user or by bad system configuration?

Example: Carat

Local instrumentation on one device, user, system configuration, ..., not sufficient for classification of power draw

App must be run under different conditions to see if (and how much) changing aspects of system improves battery.

Idea: Use collaborative approach (community) to analyze app under various conditions

Carat measures reference distribution of discharge rates during normal usage

Example: Carat

- When new app is introduced, its impact on average discharge rate is measured
- Depending on inpact to average discharge rate, app may be categorized as energy hog or bug:

Energy Hog

An app is a energy hog if its usage drains battery much faster than the average app (affecting the entire device community).

Some energy hogs may have good reasons (e.g. camera app), but classification may make user aware of energy drain.

Drain of energy hogs are unlikely to be fixed by app restarts - should run as little as possible

However: Apps using energy intensive resources (e.g. radios) are not necessarily hogs (if resources are not overused)

Example: Carat

- When new app is introduced, its impact on average discharge rate is measured
- Depending on inpact to average discharge rate, app may be categorized as energy hog or bug:

> **Energy Bug (Carat)**
>
> An app is a energy bug if it drains the battery much faster on a device than on the average on other devices.
>
> Due to the collaborative nature, detecting energy bugs over a variety of configuration (usage patterns, devices, configuration) may be easier and causes of them more isolated.

Example: Carat

- Carat app shows list of hogs and bugs

- Allows user to perform actions (kill hogs, restart bugs) and shows expected improvement by action

- App can also notify user on OS updates that improved efficiency across the community

- After 90+ days, Carat's action recommendations improved battery live by 41% for long-term users (compared to 7.9% in control group)

- 95% of the estimated time improvement (with a confidence of 95%) were correct.

Android Case Study

Introduction

- Android is one of the biggest smartphone OS
- Due to the naturity of the hardware, battery optimization is a key part of Android

  As Android is an Open Source project, we can study exactly how it works! ☺
- The complete profiling system is based on three subcomponents:

BatteryStats	Power Profile	Power Model

- Android has an On-Device Profiler with Off-Device Model Construction

  So the Model comes pre-initialized when you buy a device

  Profiling is done on a daily basis on the device

BatteryStats - Android

- BatteryStats is responsible for tracking hardware usage

  Records usage time along timestamps

  Does not directly measure energy draw from the battery
- Has two main working mechanisms:

> **Push Mechanism**
>
> Subcomponents push the component state change to the BatteryStats daemon.

> **Polling Mechanism**
>
> BatteryStats pulls information periodicallt from the `proc/` filesystem

- BatteryStats usually saves 30min of statistics in case of reboot
- It also provides statistics to requestisting services

Power Profile - Android

- BatteryStats uses Power Profile values to estimate power draw per component
- Android is installed in heterogeneous hardware. Who provides this values?

  Each Android device vendor has to fill it with custom values in Android mainline
- Android deliberately has incorrect default values to force vendors to do measurements

main/core/res/res/xml/power_profile.xml

```
<item name="screen.full.display0">0.1</item>  <!-- ~100mA -->
<item name="bluetooth.active">0.1</item> <!-- Bluetooth data transfer, ~10mA -->
<item name="bluetooth.on">0.1</item>  <!-- Bluetooth on & connectable, but not connected, ~0.1mA -->
<item name="wifi.on">0.1</item>  <!-- ~3mA -->
<item name="wifi.active">0.1</item>  <!-- WIFI data transfer, ~200mA -->
<item name="wifi.scan">0.1</item>  <!-- WIFI network scanning, ~100mA -->
<item name="audio">0.1</item> <!-- ~10mA -->
<item name="video">0.1</item> <!-- ~50mA -->
<item name="camera.flashlight">0.1</item> <!-- Avg. power for camera flash, ~160mA -->
<item name="camera.avg">0.1</item> <!-- Avg. power use of camera in standard usecases, ~550mA -->
<item name="gps.on">0.1</item> <!-- ~50mA -->
<item name="radio.active">0.1</item> <!-- ~200mA -->
<item name="radio.scanning">0.1</item> <!-- cellular radio scanning for signal, ~10mA -->
[...]
```

Power Model - Android

- With usage times & measured power values we can estimate consumption

Subcomponent/ Application	Statistical Variable	Models
Screen	Time spent at brightness level i, T_{bri-i}	$E_{Screen} = \sum_{i=1}^{N}(P_{brightness} \times T_{bri-i})$
System Idle	The total duration T_{total}; time spent when screen is on, $T_{screenOn}$	$E_{Idle} = P_{cpuIdle} \times (T_{total} - T_{screenOn})$
Radio (Cell Standby)	Time spent when signal strength is i, T_{str-i}; total time spent in scanning, T_{scan}	$E_{mobileStandby} = (\sum_{i=1}^{N} P_{strength} \times T_{str-i}) + P_{radioScan} \times T_{scan})$
Phone (Call)	Duration of a call, i, T_{call-i}	$E_{call} = \sum_{i=1}^{N}(P_{call} \times T_{call})$
Bluetooth	$T_{bluetoothOn}$, $Ping_{count}$	$E_{Bluetooth} = (P_{bluetoothOn} \times T_{bluetoothOn}) + (Ping_{count} \times P_{atCommand})$
Wi-Fi$_{App}$	Total duration an app, i, uses Wi-Fi, $T_{wifiApp-i}$; scan time for the app, $T_{wifiScan-i}$	$E_{wifiApp} = T_{wifiApp-i} \times P_{wifiOn} + T_{wifiScan-i} \times P_{wifiScan}$
Wi-Fi$_{noApps}$	Total Wi-Fi usage time $T_{wifiGlobal}$; Wi-Fi usage time by an app, i, $T_{wifiApp-i}$	$E_{wifinoApps} = (T_{wifiGlobal} - \sum_{i=1}^{N} T_{wifiApp-i}) \times P_{wifiOn}$
CPU$_{App}$	Time spent at speed, i, $T_{speed-i}$; time spent in executing app code, $T_{appCode}$; time spent to execute system code, $T_{sysCode}$	$E_{cpuApp} = \sum_{i=1}^{N} \frac{T_{speed-i}}{\sum_{i=1}^{N} T_{speed-i}} \times (T_{appCode} + T_{sysCode}) \times P_{speed-i}$
Wakelock	Wakelock ime, $T_{wakeLock}$	$E_{wakeLock} = (P_{wakeLock} \times T_{wakeLock})$
GPS	GSP usage time, T_{gps}	$E_{gps} = (T_{gps} \times P_{gps})$
Mobile Data (Byte/Sec)	Radio active time, $T_{radioActive}$	$mobileBps = (mobileData \times 1000/T_{radioActive})$
Wi-Fi Data (Byte/Sec)	Wi-Fi active time, $T_{wifiActive}$	$wifiBps = (wifiData \times 1000/T_{wifiActive})$
Average Energy Cost per Byte		$E_{byte} = (\frac{P_{wifiActive}}{wifiBps} \times wifiData + \frac{P_{radioActive}}{mobileBps} \times mobileData)/(wifiData + mobileData)$
App		$E_{App} = E_{cpuApp} + E_{wakeLock} + E_{wifiApp} + E_{gps} + (tcpBytesReceived + tcpBytesSent) \times E_{byte}$

Profiling Android Applications

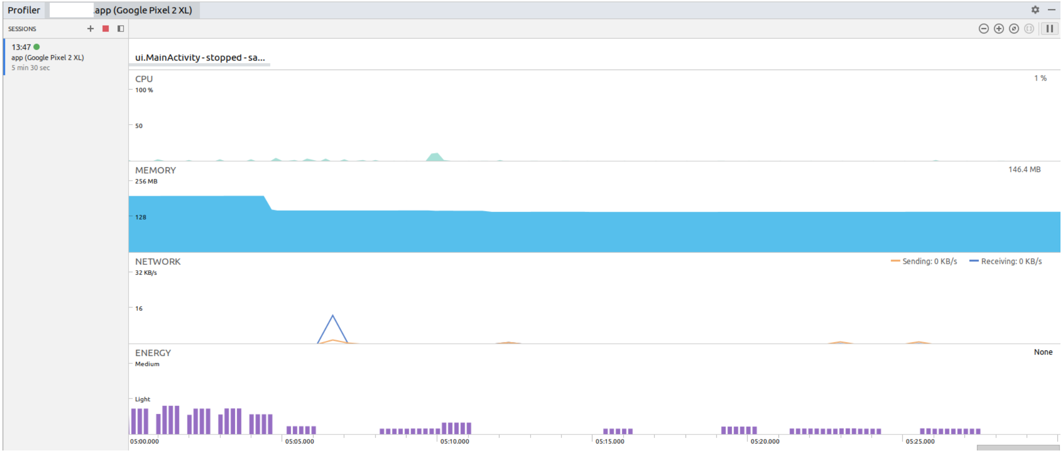
- Android provides a mechanism to access power estimation mechanisms

```
>_ adb shell dumpsys batterystats -checkin
```

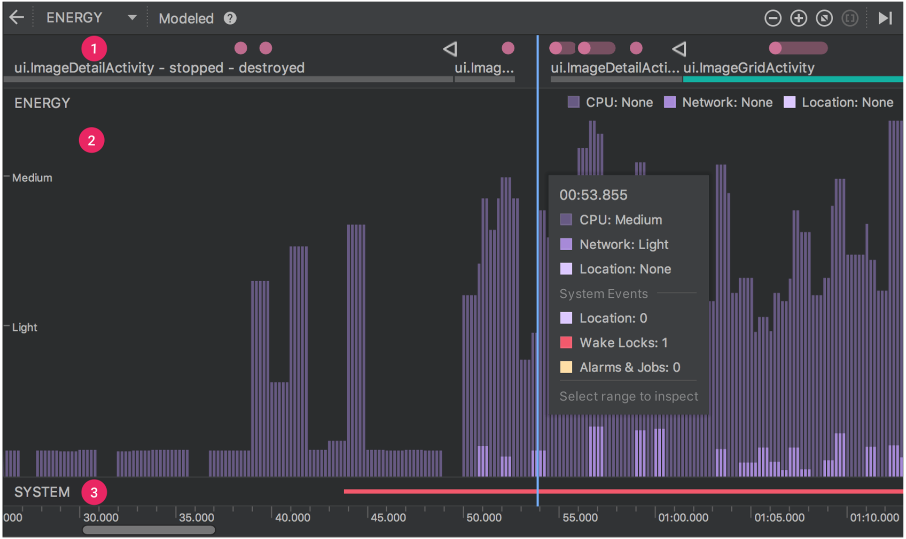
```
9,0,i,vers,11,116,K,L
9,0,i,uid,1000,android
9,0,i,uid,1000,com.android.providers.settings
9,0,i,uid,1000,com.android.inputdevices
9,0,i,uid,1000,com.android.server.telecom
...
9,0,i,dsd,1820451,97,s-,p-
9,0,i,dsd,3517481,98,s-,p-
9,0,l,bt,0,8548446,1000983,8566645,1019182,1418672206045,8541652,994188
9,0,l,gn,0,0,666932,495312,0,0,2104,1444
9,0,l,m,6794,0,8548446,8548446,0,0,0,666932,495312,0,697728,0,0,0,5797,0,0
```

- This raw information can be plotted via different tools

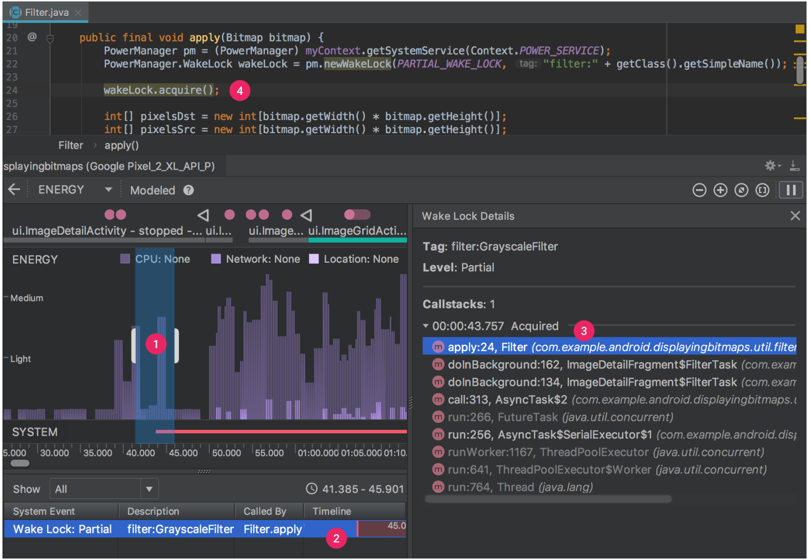
Android Energy Profiler



Android Energy Profiler



Android Energy Profiler



Battery Historian

Battery Historian

App Selection

Sort apps by

Name

Choose an application

Tables

- System Stats
  - Aggregated Checkin Stats
  - Device's Power Estimates
  - Userspace Wakelocks
  - SyncManager Syncs
  - JobScheduler Jobs
  - CPU Usage By App
  - Mobile Radio Activity Per App
  - Mobile Traffic Per App
  - WiFi Scan Activity Per App
  - WiFi Full Lock Activity Per App
  - WiFi Traffic Per App
  - Kernel Wakesources
  - Kernel Wakeup Reasons
  - App Wakeup Alarms
  - App ANRs and Crashes
  - GPS Use By App
  - Time Spent In Each App State

System Stats | History Stats | App Stats

Duration / Realtime: 42m10.598s

Aggregated Checkin Stats:

☐ Show metrics with 0 values.

Copy

Metric	Value
Screen Off Discharge Rate (%/hr)	6.55 (Discharged: 2%)
Screen On Discharge Rate (%/hr)	15.10 (Discharged: 6%)
Screen On Time	23m50.612s
Screen Off Uptime	8m39.306s
Userspace Wakelock Time	6m20.288s
Sync Activity	5m46.689s (71 times)
JobScheduler Activity	6m13.044s (106 times)
App Wakeup Alarms	219 times
CPU Usage	22m45.447s user time, 16m5.655s system time
Kernel Overhead Time	2m19.018s
Kernel Wakelocks	8m8.001s (7908 times)
Wakeup Reasons	1m23.864s (185 times)
Mobile KBs/hr	40622.81
WiFi KBs/hr	716.43
Total WiFi Scan Activity	6m21.74s (967 times)
Total WiFi Full Lock Activity	5m30.855s
Mobile Active Time	27m11.858s
Signal Scanning Time	31.215s
Full Wakelock Time	12.369s
Interactive Time	23m47.619s
Total GPS Use	10m28.921s (15 times)
Wifi Power Usage	0.05%/hr, 0.04% total

Battery Historian

App Selection

Sort apps by

Name

com.google.android.youtube (Uid: 10116)

Tables

- System Stats
- History Stats
- App Stats
  - Misc Summary
  - Network Information
  - Wakelocks
  - Process info
  - Sensor Use

System Stats | History Stats | App Stats

Copy

Application	com.google.android.youtube
Version Name	12.14.12
Version Code	121412644
UID	10116
Device estimated power use	0.02%
Foreground	2 times over 3s 410ms
CPU user time	4s 920ms
CPU system time	683ms
Device estimated power use due to CPU usage	0.00%

Network Information:

Search: Copy

Mobile data transferred	99.41 KB total (71.18 KB received, 28.23 KB transmitted)
Wifi data transferred	0.00 bytes total (0.00 bytes received, 0.00 bytes transmitted)
Mobile packets transferred	226 total (109 received, 117 transmitted)
Wifi packets transferred	0 total (0 received, 0 transmitted)
Mobile active time	14s 890.63ms
Mobile active count	2
Modem idle time	0s
Modem transfer time	8s 745ms total (8s 410ms receiving, 335ms transmitting)

Wakelocks:

Process info:

Sensor Use:

Application Developer Safeguards

- Android restricts code that could waste power inefficiently

  You can't have services running permanently

  They even patch bypasses to this restriction (recent Android 14 watchdog technique)

  You have to use their APIs, which are implemented properly in a power-conscious way

- Android restricts code when device is unused: Project Doze

  During sleep state in Doze:

  - Apps can't access the internet
  - App wakeloks are ignored
  - WiFi Scans can't be done
  - `SyncAdapter` and `JobScheduler` is deferred

  Doze exits sleep state on: interaction, device movement, screen on, inminent alarm

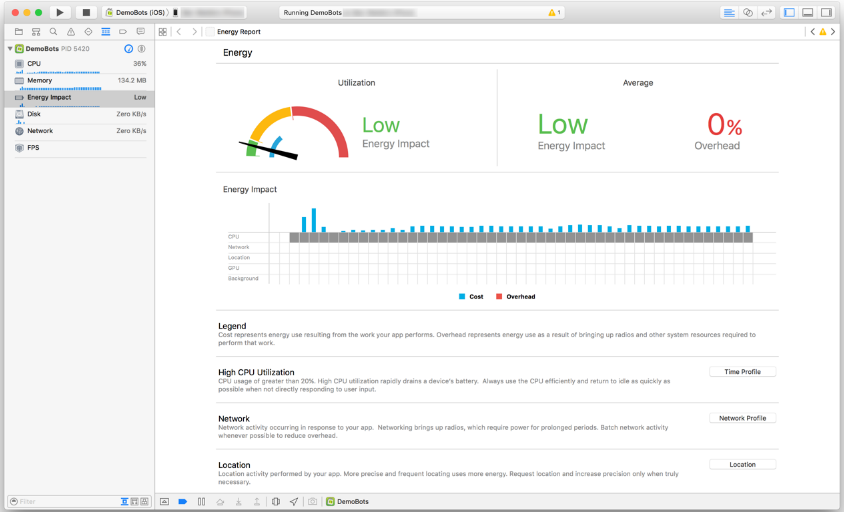
iOS Case Study

Profiling iOS Applications

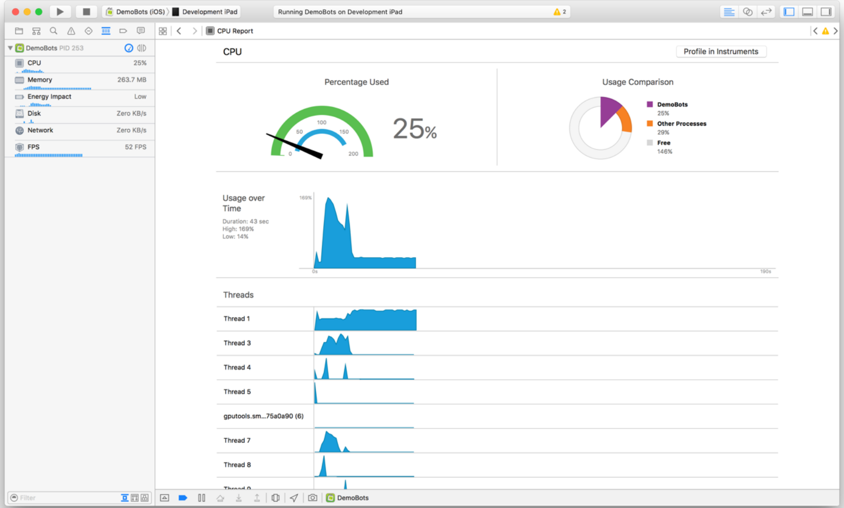
- Apple provides an official mechanism to measure energy on `iOS` applications
  - Per-application, one can study power usage per subsystem/component
  - Apple also assigns an overall usage: Low, Medium, High
- Measurements can be done through XCode[1]
- </> Developers can use this information to optimize power in their apps
- We’ve also found papers & articles regarding API usage recommendations

[1] https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/MonitorEnergyWithXcode.html

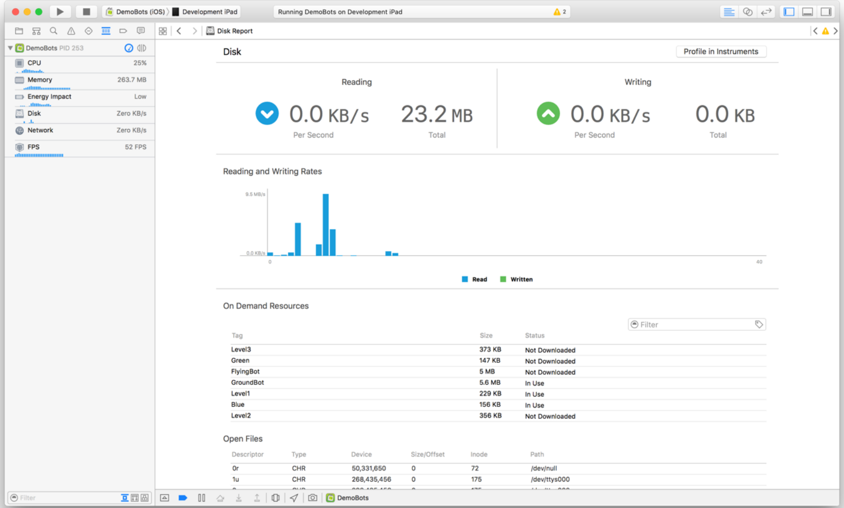
Profiling iOS Applications



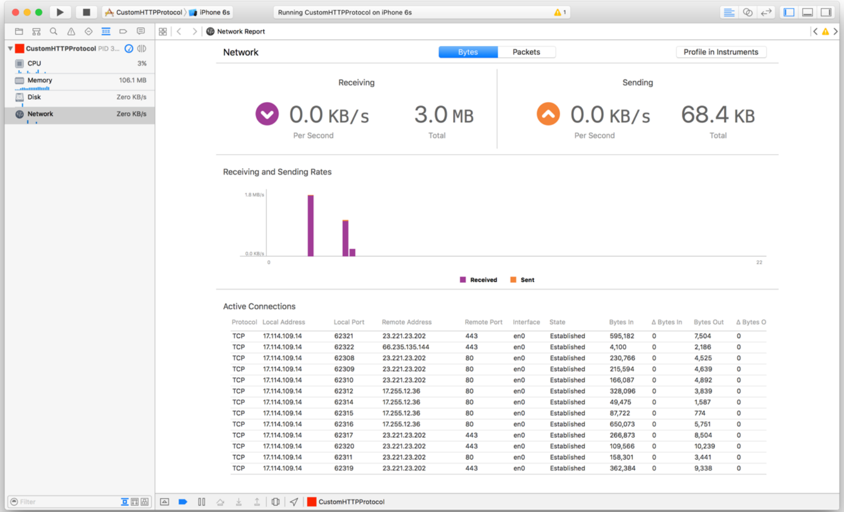
Profiling iOS Applications



Profiling iOS Applications



Profiling iOS Applications



Models Used

- ❓ Which models does Apple use then? How do they calculate them?
- 🔒 `iOS` is closed source, Apple is reticent on its internals
- ☹ Sadly we can't find Android's equivalent information for `iOS`

Summary

Recap & Takeaways

- Due to limitations, power optimization is a keystone in mobile devices
- Estimating per-module power draw is not trivial and requires modeling
- Vendor ships software components with pre-trained power models
- When developing applications, we can make use of energy profilers
- Operating Systems are very strict on energy usage

Bibliography

[1] Niels Brouwers, Marco Zuniga, and Koen Langendoen. "NEAT: A Novel Energy Analysis Toolkit for Free-Roaming Smartphones". In: Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems. SenSys '14. Memphis, Tennessee: Association for Computing Machinery, 2014, pp. 16–30. ISBN: 9781450331432. DOI: 10.1145/2668332.2668337. URL: https://doi.org/10.1145/2668332.2668337.

[2] Christian Clemm et al. " „Market Trends in Smartphone Design and Reliability Testing " ". In: Fraunhofer-Institut für Zuverlässigkeit und Mikrointegration (2020), pp. 171–178.

[3] Mohammad Ashraful Hoque et al. "Modeling, Profiling, and Debugging the Energy Consumption of Mobile Devices". In: ACM Comput. Surv. 48.3 (Dec. 2015). ISSN: 0360-0300. DOI: 10.1145/2840723. URL: https://doi.org/10.1145/2840723.

[4] Adam J. Oliner et al. "Carat: Collaborative Energy Diagnosis for Mobile Devices". In: Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems. SenSys '13. Roma, Italy: Association for Computing Machinery, 2013. ISBN: 9781450320276. DOI: 10.1145/2517351.2517354. URL: https://doi.org/10.1145/2517351.2517354.

[5] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. "Bootstrapping Energy Debugging on Smartphones: A First Look at Energy Bugs in Mobile Devices". In: ACM HotNets. Jan. 2011. URL: https://www.microsoft.com/en-us/research/publication/bootstrapping-energy-debugging-on-smartphones-a-first-look-at-energy-bugs-in-mobile-devices/.

Power Aware Computing

Mechanisms for Power Draw Analysis on Mobile Devices

>_ PROD v1.3 ✔

Ernesto Martínez García [1]
me@ecomaikgolf.com

Marcell Haritopoulos [2]
marcell.haritopoulos@student.tugraz.at

Graz University of Technology

Power Aware Computing LU SS23

23rd of November 2023

SLIDES & REPORT

ls.ecomaikgolf.com/slides/power-aware-comp