# The Model Is Not Enough: Understanding Energy Consumption in Mobile Devices

James Bornholt
Australian National University
u4842199@anu.edu.au

Todd Mytkowicz
Microsoft Research
toddm@microsoft.com

Kathryn S. McKinley
Microsoft Research
mckinley@microsoft.com

## 1. Introduction

Although battery life has always constrained embedded and mobile hardware developers, the rise of smart phones and tablets has also made energy a fundamental concern of software developers. On the desktop, software developers generally ignored energy, but in the mobile environment, battery life is critical to the user experience. Just as developers use performance profiling tools, they now need energy profiling tools to understand how and why their software consumes energy.

The inconvenience, cost, and complexity of external power measurement hardware and the inaccuracy of on-board power sensors on older phones [1] motivated researchers to create power modelling tools. Power modelling uses utilization metrics to estimate power draw based on previously measured correlations between the metrics and power.

We show that the on-board power sensor is now accurate on a Windows Phone 7.5 device running on a SnapDragon MSM8660. Compared to external measurement hardware, the on-board "fuel gauge" is accurate to within 2% of total energy consumption. We thus modify the Windows Phone 7.5 OS to sample power without external hardware, sample the application call stack to correlate energy consumption with code, and examine power traces from two weeks of normal use. These traces illustrate behavior where modelling alone is not sufficient to understand the energy consumption of a mobile device. For example, we observe inter-day variations in base power draw as the battery discharges, an effect that to our knowledge is not captured by existing modelling work.

This work recommends that a hybrid approach will improve the accuracy of energy profiles, and that direct measurements will significantly improve the accessibility of fine-grained energy information in both testing and deployment. Armed with easy-to-use energy analysis tools, hardware designers, OS developers, and third party application developers will be better equipped to understand and optimize the energy behavior of mobile code.

## 2. Related Work

Early energy modelling research used power measurements of executing each machine instruction [4, 5]. These methods do not extend well to modelling the power draw of other non-CPU components, which constitute two thirds or more of energy consumption on mobile devices.

For mobile devices, recent models use linear regression trained on energy profiles of the entire device gathered from scenarios that stress each device component [3]. This model is accurate compared to external measurements on short-duration test runs. This approach, however, cannot address the tail power state problem, where components improve responsiveness by waiting in a higher power state for more work to arrive (see Section 4.1).

To provide fine-grained energy accounting, Pathak et al. [2] introduce a finite state machine that models component power draw by tracing system calls. For example, a `read` system call transitions the flash storage component into a higher power state. This approach is promising for attributing energy consumption to code. However, it does not model power draw of key device components, such as the screen or application CPU draw, and so cannot present a complete picture of the device's energy usage.

Dong and Zhong [1] overcome these problems with a hybrid approach. They create models on-the-fly using the on-board battery sensor. They use low-frequency samples from the sensor to track accuracy and trigger model reconstruction if the variation is too high. This method potentially accounts for variation in base power draw, but with a significant cost or latency. However, it does not address the tail power state problem.

## 3. Measuring energy consumption

External measurement hardware, such as the Power Monitor with which we compare in this paper, accurately measures power draw from a phone's battery. These tools, however, are relatively expensive ($750, which is more than the phone itself) and limit phone mobility, restricting real world testing.

On modern mobile devices, the "fuel gauge" (FG) provides accurate readings of battery voltage and instantaneous current draw for displaying remaining battery life. To validate the FG's accuracy, we executed benchmarks on a HTC (MSM8660 processor) device running Windows Phone 7.5, simultaneously capturing power measurements from the FG and an external Power Monitor. The FG was accurate to within 2%$\pm$0.02 of the Power Monitor. We suggest this accuracy is enough to replace external hardware as the source of power measurements.

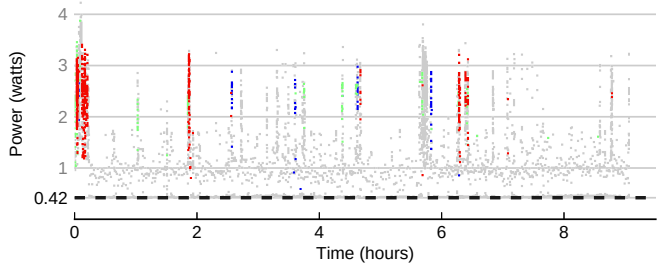
## 4. Power modelling

The main challenge in on-board energy profiling is accurately attributing the energy consumed by particular applications and methods. The traditional approach to this problem has been *modelling*, which can both produce power readings and attribute them to code entities.

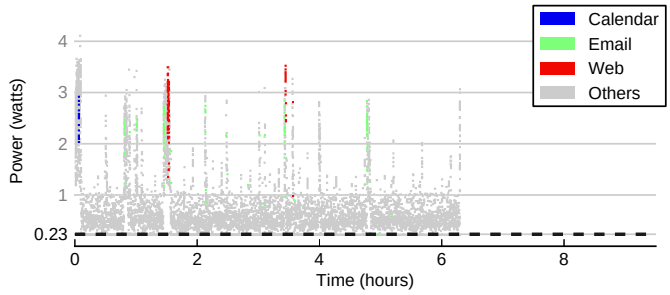
### 4.1 Tail power states

Even with a model, *tail power states* complicate energy profiling. To provide responsiveness, many components (e.g., radio, GPS) continue to draw high power after use. For example, a 3G radio may remain in a higher "tail" power state for up to seven seconds after use. This tail power state complicates energy attribution: the download has completed, the code has moved on, and the application may no longer be running, but power is still drawn. Further, if several applications use a component, which one should be charged for the tail power state?

Pathak et al. [2] model tail power states with their system call model, record the calling context of each system call, and assign the tail power to the last calling context that used the device.




(a) Day 1: Idle power 0.42 W




(b) Day 2: Idle power 0.23 W

**Figure 1.** Two days of power usage with day-to-day variation. Points are coloured by active application.

### 4.2 Shortcomings of modelling

There are two main drawbacks to modelling:

1. Accuracy is limited by the training environment and components modelled (e.g., CPU and screen are often missing). Because the mobile environment is so diverse (hardware models, cellular networks, etc.), modelling has the potential to be very inaccurate.
2. The trade-off between latency, cost, and portability essentially limits models to testing. Shipping system call traces to a server introduces long latency, but running models on-board has high overhead and can require external hardware, impeding portability.

Our experiments also show a significant day-to-day variation in base power draw when the phone is "idle." Figure 1 plots two days of power readings, revealing a variation of 0.19 W in base power draw between two consecutive days of usage (the dashed line at base of each figure). A point $(x, y)$ on this graph plots the power in watts ($y$) as a function of a day's usage (time on the $x$ axis). Day 1's base draw is 0.42 W while Day 2's is 0.23 W, which is a significant difference in total energy consumption over several hours. Unless this variation is observed in the lab, a model is unlikely to predict this day-to-day variation.

## 5. A hybrid approach

We advocate a hybrid approach to accurately attribute energy consumption to the code that uses it: accurate battery sensors (like the FG) to gather online power readings, and system call modelling to handle components with tail states.

This approach has several benefits. First, it ensures that an energy profile always shows a complete picture of system energy consumption. By always capturing whole system power readings, components acting in ways unanticipated by the model do not go unobserved. While precise attribution is challenging, the recent applications and their call stacks will help identify problems, particularly if applications are sampled over many days. Real measurements at the very least identify power anomalies, and that the model may need correction. Second, a hybrid approach deals with tail power states in a way measurement alone cannot. For example, it can accurately assign the energy usage of networking hardware to the code making network calls, providing a more actionable energy profile to a developer. Finally, if we use on-board sensors, we can avoid the need for expensive external hardware, improving the accessibility of energy information to developers.

This approach also opens a new field of potential optimization at the OS level. With completely online power measurements, the OS may monitor battery life performance at a fine-grain in real time, and perform optimization to achieve a battery life goal. For example, if an alarm is set for 8 hours in the future and projecting current energy consumption indicates that the battery will not last that long, the OS can optimize components to meet this power goal. This approach improves over the current practice, in which the OS takes drastic measures when the battery is very low and it is too late to recover. Guided by the component attribution possible with modelling, the OS may determine that the GPS is consuming too much energy and tune it to use less energy by providing less accurate positions. Gathering the data must have low overhead (less than 4% in our testing), such that it does not contribute to the problem. The OS needs fine-grained real-time low cost power data to make these types of optimizations. Our tool provides such data.

## 6. Conclusion

Mobile devices are placing energy efficiency in the hands of software developers. Unfortunately, power modelling alone cannot identify significant power variations, nor is it practical in deployment. We show that on-board fine-grained power measurements of the fuel gauge are now both accurate and low overhead.

We advocate a hybrid approach to power measurement: combining the best aspects of both modelling and measurement to produce accurate and actionable energy information for developers. Using such tools, developers have the potential to understand and optimize the energy behavior of their code. Operating system developers have the potential to guide real-time optimization in the interests of battery life, a significant advancement over the current state of power optimization. These advances are critical to the future of mobile software, as developers at all levels come to terms with their new-found responsibility for energy efficiency.

## References

[1] M. Dong and L. Zhong. 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, pages 335–348, June 2011.

[2] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 153–168, Apr. 2011.

[3] A. Shye, B. Scholbrock, and G. Memik. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 168–178, Dec. 2009.

[4] P. Stanley-Marbell and M. Hsiao. Fast, flexible, cycle-accurate energy estimation. In *Proceedings of the 2001 international symposium on Low power electronics and design*, ISLPED '01, pages 141–146, 2001.

[5] V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee. Instruction level power analysis and optimization of software. In *Proceedings of the 9th International Conference on VLSI Design: VLSI in Mobile Communication*, VLSID '96, pages 326–328, 1996.

# The Model Is Not Enough: Understanding Energy Consumption in Mobile Devices

James Bornholt — Australian National University — u4842199@anu.edu.au
Todd Mytkowicz — Microsoft Research — toddm@microsoft.com
Kathryn S. McKinley — Microsoft Research — mckinley@microsoft.com

## Why understand energy?

Smart phones and tablets force software developers to focus on battery life.

Developers already optimise performance using profilers. Our goal is to build an energy profiler.

## Why is energy profiling difficult?

The two ways to calculate energy—hardware power meters and software modelling—have drawbacks.

Attribution of energy to code is made difficult by tail power states, which shift the blame.

### Tail power states: who's to blame?

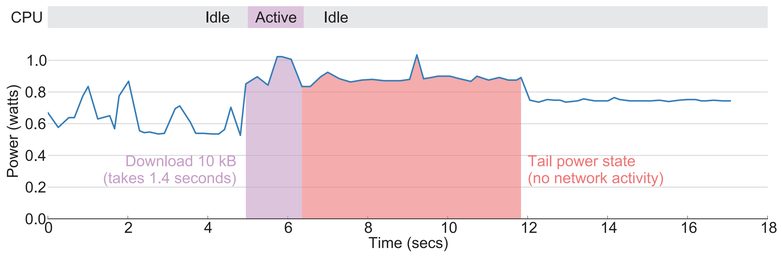



**Figure 1** The 3G radio's tail power state consumes energy while the CPU is idle.

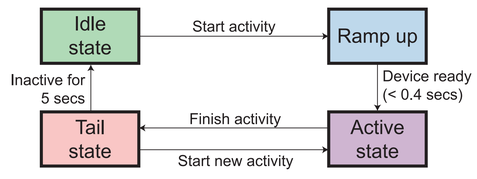



**Figure 2** Tail power states avoid returning to the ramp up state, therefore reducing latency.

## A deeper look: energy profiling

**Why understand energy?**

- Software and energy interact in subtle and non-trivial ways – for example, grouping of network requests to avoid thrashing the network hardware
- Energy bugs present serious usability problems, because users cannot easily identify them until it is too late (i.e. the battery is empty)

**Why is energy profiling difficult?**

- Hardware meters are often expensive and bulky
- Models are specific to their training environment
- Tail power states reduce ramp-up latency by allowing components to remain powered on after last use
- So it's wrong to attribute current power draw to the currently executing code – the real culprit may be long gone!

## Power modelling

Modelling uses metrics such as CPU and network usage to extrapolate power draw, but this has issues for profiling use.

Recent work uses system call tracing to handle tail power states, but other issues remain.

## Power measurement

Hardware power meters are common for power measurement, but these are impractical for most developers.

Our work shows the onboard "fuel gauge" battery sensor is accurate to within 2% of external meters, but measurement alone cannot address tail power states.

### Modelling misses important variation

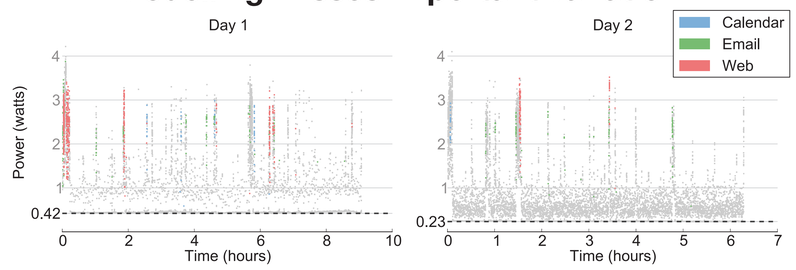



**Figure 3** Modelling misses a base power variation, worth 30% of battery life over 8 hours.

## A deeper look: modelling and measurement

**Power modelling**

- The raw numbers are often quite good: errors of below 10%
- But the models:
  1. Do not address tail power states
  2. Are specific to the lab environment they are trained in
  3. Cannot detect the unexplained power variation in Figure 3
- In system call tracing, finite state machines model each component's power states, using system calls to transition the machine
- This addresses the tail power state problem, but suffers the other drawbacks of modelling, and also cannot yet capture important components like the screen

**Power measurement**

- External meters are expensive and bulky, making them inaccessible and difficult to use
- The fuel gauge typically measures battery capacity, but modern sensors provide instantaneous power draw
- When testing an HTC Windows Phone, the onboard sensor was accurate to within 2% of total energy compared to the power meter
- The overhead of sampling the fuel gauge at 5 Hz was less than 4%.
- This means the fuel gauge is accurate enough to replace the external power meter for many uses, but may not be appropriate for very high frequency sampling
  - The power meter we used samples at 5000 Hz

## A hybrid approach

Based on our results, we advocate a hybrid approach to energy profiling.

This approach makes energy profiling more accurate, more actionable, and more accessible.

## What do we get from this data?

Energy profiles let developers isolate poor energy usage in their code.

Energy profiles open a new field of potential online OS-level energy optimisations.

### Energy profiles identify real bugs

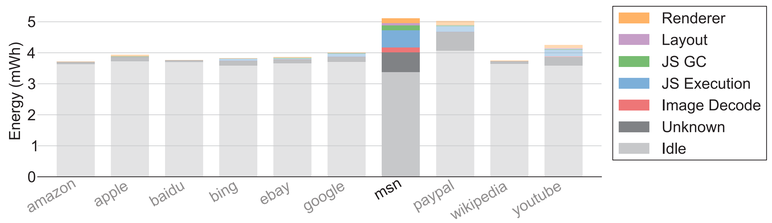



**Figure 4** An energy bug in the MSN website, seen and diagnosed by an energy profiler.

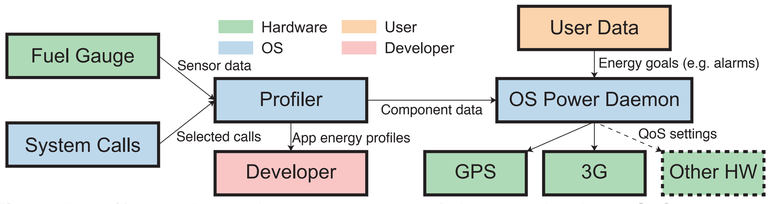



**Figure 5** Profiles can be used to meet energy goals by tuning hardware QoS settings.

## A deeper look: hybrid energy profiles

**A hybrid approach**

- Accurate onboard sensors for online, whole-system power readings, and system call modelling to handle tail power states
- Energy profiles will always show a complete picture of system energy use, rather than omitting unmodelled components, making profiles more accurate
- Tail power states are handled, making profiles more actionable
- By removing the need for hardware, even for training, profiles are more accessible

**What do we get from this data?**

- In Figure 4 we see MSN as a clear outlier in energy use
- The profile identifies excessive image preloading as the cause of this inefficiency
- The OS can use online profiles to achieve battery life goals; for example, tuning energy use to ensure a scheduled alarm goes off
- Guided by component attribution, this tuning can target specific energy users like the GPS, and trade accuracy or speed for battery life

Australian National University

Microsoft Research