The Case for Device Power States

Marcelo Martins and Rodrigo Fonseca

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-11-03
October 2011

The Case for Device Power States

Marcelo Martins
Brown University

Rodrigo Fonseca
Brown University

Abstract

Understanding power draw in mobile devices has received enormous attention, as energy is *the* limiting resource in these platforms. In the absence of hardware measurements we must resort to models that predict power usage based on events seen by the OS. The accuracy of these models depends on the type of model and on the set of events used as inputs. By looking in detail at the power draw of a wireless card in a laptop running Linux, in this paper we assess the quality of the models one can obtain using events from different levels of the software stack.

In recent work, Pathak *et al.* advocated the use of system-call tracing, together with an FSM model of components, for predicting power in smartphones. While their technique presents significant advantages over previous counter-based models, we show two concrete examples in which using events at the system-call layer will lose significant accuracy compared to models using events from the device-driver layer. We examine the tradeoffs of using events from the two layers, and argue that device drivers should export a narrow interface that exposes underlying power states to enable simple, accurate, and universal power models by the operating system.

1 Introduction

Energy is the most limiting resource in today's sophisticated mobile devices. Given that battery technology progresses much slower than the multimedia and networking applications' need for energy, it is essential that these devices make efficient use of their energy resources. Understanding this usage enables developers to tell which hardware components, acting on behalf of which software activities, are spending how much energy. Such capability is key to productive whole-system energy optimization, in the same way that widespread use of CPU and memory profiling enables performance optimization.

Correspondingly, there is a rich body of literature on measuring (*e.g.*, [9, 10, 17]), modeling (*e.g.*, [3, 8, 13]), and managing (*e.g.*, [15, 16, 18]) energy usage, on platforms ranging from sensor networks, to mobile phones, laptops, and servers.

Ideally, we should have energy-measurement hardware in all platforms, preferably instrumenting each individual subcomponent. While we have had significant progress in that front, most platforms today have very rudimentary, if any, energy-measurement capabilities. In these cases, we must resort to estimating energy usage using pre-trained models. In §2 we discuss the extensive literature on power modeling. In general, power models take as inputs events visible to the OS to predict power draw. Two important categories are those models based on event counts, and those that model hardware components as finite-state machines (FSMs.) Apart from the model type, prediction performance depends crucially on the input events, which can be collected at different layers of the software stack.

Selecting the input events to the power model has significant tradeoffs. The system-call layer is much more easily instrumented, and Pathak *et al.* [14] have shown very good results with an FSM-based model using system-call-level events. However, as we show in our evaluation in §4, there are instances in which power changes are *not* reflected at the system-call layer, causing arbitrary loss of accuracy.

On the other hand, the device driver is the OS component closest to the hardware, and with the po-

tential to enable the most accurate models. Device drivers are, however, numerous and diverse, are often proprietary, disconnected from user-level operations, and may offer more detail than necessary. As we show in Section 4, device drivers have the potential to allow much simpler and more precise models, and that, despite their diversity, could do this by providing a very simple and universal interface for exposing their power states to the OS.

The central goal in this paper is to assess the quality of power models one can obtain using events from different layers of the software stack.

By closely measuring the power draw of a wireless card on a Linux laptop, correlated with power-related events collected in synchrony at the system-call and device-driver layers of the software stack, we examine the tradeoffs involved in modeling power with the two sets of events. We show two examples where a system-call-level model loses significant accuracy due to power-related events that are not reflected in that layer. Coupled with the fact that the device-driver model is simpler, and does not suffer from the same anomalies, we argue that device drivers should expose the power state of their underlying hardware through a narrow interface, to enable simple, accurate, and universal power modeling by the operating system.

2 Background

There is an extensive body of work in trying to measure and model energy usage in mobile devices. In this work we focus on modeling, the only option when detailed instrumentation is not available on a given platform.

Counter-based models: Bellosa [2] pioneered the use of linear models based on event *counts*, such as those provided by performance counters, by noticing strong correlations between specific counters and CPU power. Several other works followed using similar approaches, varying in granularity and in the scope of events used as inputs. Contreras and Martonosi [6] used performance counters and a linear model for online power estimation of CPU and memory in an XScale platform, reporting average relative errors of 4%. Bircher and John [3] extended a similar model to additionally account for energy used by the chipset, I/O, and disk subsystems, and reported errors of less than 9% for all subsystems. Mantis [8] used measurements of CPU and disk utilization in addition to performance counters for whole-system power profiling, and achieved errors, for two platforms, within 15%. More recently, McCullough *et al.* [13] compared several linear and non-linear regression models based on performance counters, and found that despite achieving good accuracy, these models perform worse for individual hardware components that change power states with no signals to the OS.

Finite State Machine models: An implicit assumption of the counter-based models is that each occurence of a event implies some energy expenditure. This assumption breaks down if the marginal energy cost of an event is low compared to the active power draw of a subcomponent. For example, in the CC2420 radio [11], used in wireless sensor networks, the power for transmitting a packet is almost the same as the listening power, and that the number of transmissions is a very poor estimator of energy usage. Another problem with counter-based models is that they require sampling, and suffer from the inherent tradeoff between overhead and agility when selecting a sampling rate.

An alternative approach that does not rely on this assumption is to model the hardware subcomponents as finite state machines (FSMs), and use events to trigger transitions. These models can achieve higher accuracy than their counter-based counterparts, as they arguably model devices as what they really are: state machines. FSM models can more readily accomodate domain knowledge, such as notions of batching and timeouts, and can account for events or conditions which cause state transitions, but do not, on their own, incur energy usage.

In the realm of wireless sensor networks, Dunkels *et al.* [7] modeled hardware components as two-state FSMs, and used a linear model based on the time each component spent on each state, multiplied by the previously measured power draw of these states, to predict energy usage online. Kellner [12] also

used a state machine to model hardware components in a sensor-network platform. The states are those that have distinguishable power draws, and they also assign a fixed amount of energy per transition. Quanto [10] breaks the hardware subcomponents into logical units that can be on or off and instruments device drivers to inform the OS of the state of these units. It then splits the measured full-platform energy among the units based on a linear model on the time each unit was on. Cignetti *et al.* [5] also used a set of FSMs to model components in the Palm palmtop computer, and suggested the use of system calls to trigger transitions in the states of the model. Using a similar model, but in the context of modern mobile phones, Pathak *et al.* [14] showed that an FSM model using input from the system-call layer could significantly outperform counter-based models. They attribute the improved performance to the presence of tail power states, system calls that change power state but imply no utilization, and to components that have no quantitative utilization, but have distinct power states.

3 Comparing Different Layers

Our goal in this paper is to assess the quality of the power models we can obtain using input events from different layers.

Similarly to [14], we used FSM-based models using input events from two different layers: the system-call layer and the device-driver layer. To make our discussion concrete, we describe the two models of the Atheros AR928X 802.11n wireless card, shown in Figures 1 and 2, based on the events we collected on an instrumented Linux kernel. We describe our experimental setup in more detail in §4.

Each state in the FSM is a state with distinguishable power draw in the hardware device. State transitions can be triggered by specific events and by timeouts since specific events. We do not consider cumulative counts of specific events in the past, a feature in [14], since we did not see any correlation between the number of counts and changes on the power curve in our radio experiments.

The two FSM models we used for the two layers are simpler than the ones described in [14], as the only events that changed the power state of the wireless card we examine here are the ones listed in Figures 1 and 2.

Figure 1 shows the FSM for the system-call layer. Here we do not have access to explicit events that signal actual state transitions, and infer these from the behavior of the power curve around sending and receiving system calls. For example, for a send system call, the card goes into its active power mode, transmits a number of packets proportional to the number of bytes written, and stays in the active (tail) state for a timeout interval. If a new write call arrives before the timeout interval, the timeout interval is reset and starts again.

A particular deficiency of this model is estimating a general value for the timeout interval. Our experience shows that depending on the packet size, the value of this interval varies: in the case of transmissions, it is greater for smaller packets, while for receptions it does not vary much. Even worse, the timeout proportionality is not linear. After measuring the timeout value with message sizes ranging from 1B to 10KB, we decided to take its average as the general value. Also, as we show in §4.2, hidden-beaconing activity interferes with the timeout value, sometimes delaying the transition to sleep with no apparent cause at the socket layer.

Figure 2 shows the FSM for the device-driver level. Here we have access to more events, notably the *Active* and *Sleep* events, which directly signal when the card switches its power state. Building the model using this FSM is considerably simpler, as we do not have to estimate the timeout value from several experiments. All transitions are explicitly stated by the device driver. More importantly, activities that are not visible in upper layers of the communication stack, but are energy-relevant, are now exposed. For instance, management frames (*e.g.*, AP beacons) are transceived in a lower power level than data frames. Other energy-relevant activities include channel scanning, ICMP- and ARP-packet transmission/reception. All of these activities can be described in terms of the *Data Rx* and *Data Tx* events.

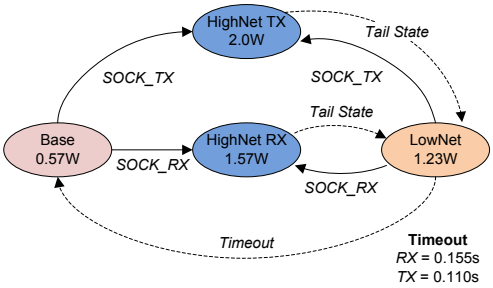


Figure 1: FSM for the system-call level. Dashed lines are transitions with no explicit events, inferred from the number of bytes transmitted or received, or from timeouts. Tail-state duration is a function of the packet size.

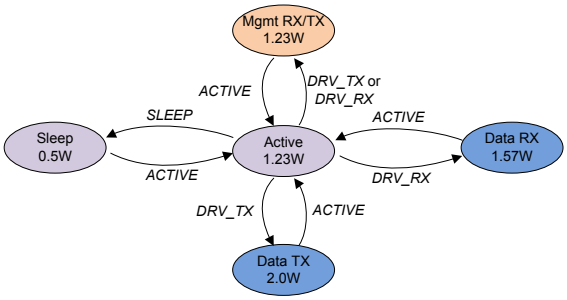


Figure 2: FSM for the driver level. Explicit state-transition events make the model simpler and more accurate.

4 Evaluation

In this section we compare the different models of power draw of the wireless card in a laptop, using events from two different layers: the system-call and the device-driver layers.

Experimental Setup

We used an Acer AspireONE D250 netbook running Ubuntu Linux "Maverick Meerkat" (10.10), with kernel version 2.6.35.10. Attached to the netbook is an Atheros AR928X 802.11n wireless card running in power-save mode, associated with a Linksys 802.11n WRT54GL router. For collecting kernel-event data, we used SystemTap [1] to instrument relevant points in the different layers of the networking stack, from the socket-related system calls to the *ath9k* driver of the wireless card.

For the ground-truth power data, we used a Mini-PCI-Express bus extender with $5m\Omega$ sense resistors (1% tolerance) to remove the wireless card from inside of the laptop case, and expose the power lines that feed the card. We measured the voltage across these sense resistors by sampling at a frequency of $100KHz$ with an NI USB-6218 DAQ. During the measurements we removed the battery from the computer to avoid interference from charging cycles. We applied a lowpass IIR filter to the voltage, and converted the measured voltage drop and given resistance to current using Ohm's law.

Note that our method of measuring power draw is not particularly novel or the only way to achieve this. In particular, other works (*e.g.* [14, 10] have shown that by properly isolating the workload one can obtain the variations in power draw by only measuring the global power draw of the platform.

Our final data consists of two time series: one of kernel events, timestamped with the kernel's internal clock, and one of power samples, timestamped with the DAQ's internal clock. We synchronized both time bases in post processing by always surrounding experiments with a small series of ping packets, which showed a very distinctive pattern in the power curve corresponding to a stable sequence of kernel events.

4.1 Noisy channels

Our first experiment involves association with a noisy communication channel. This scenario is common to many users of mobile devices connected to the Internet. Lack of proximity to access points, environmental conditions and antenna irregularities are all possible factors that lead to communication disturbance. As a result, channel quality degrades, and the wireless interface is forced to increase its transmission power to receive packets. Nonetheless, there are situations in which such efforts are not enough: Even if a packet arrives at its destination, its payload could

have been altered and there is no other option but to discard it. Still, the wireless radio spends energy to receive such packets, although the socket API is not notified about it since lower layers decided to refuse it.

4.2 Unsolicited packets

Another possible source of energy-estimation errors is the case of unsolicited packets. Parthak *et al.* assume that all packets leaving and entering an WiFi device are expected by applications, which is not necessarily true, especially for the case of packet receptions. An WiFi device makes no distinction whether a packet is destined to an existing application or not. As long as the MAC address header is correct, the packet is accepted and uploaded to upper layers. Associating packets to applications is the responsibility of the network layer, which verifies the requested port to a given socket. If the requested port does not belong to the map of open connections on the destination, the packet is discarded. Examples of such scenario include port scanning, unwanted UDP connections, broadcast and multicast.

It is important to notice here that, once again, the socket API is not notified of any packet reception, although the radio draws power to receive it. To show the effectiveness of the device-driver-based model for this scenario we run a microbenchmark consisting of two phases: (1) a small period of idle listening, and (2) a fifteen-second period of one expected UDP stream and one unexpected UDP stream flowing simultaneously.

We used an Acer AspireONE D250 netbook running Ubuntu Linux "Maverick Meerkat" (10.10), with kernel version 2.6.35.10. Attached to the netbook is an Atheros AR928X 802.11n wireless card running in power-save mode, associated with a Linksys 802.11n WRT54GL router. For collecting kernel-event data, we used SystemTap [1] to instrument various points in the different layers of the networking stack, from the socket-related system calls to the *ath9k* driver of the wireless card.

For the ground-truth power data, we used a Mini-PCI-Express bus extender with $5m\Omega$ sense resistors (1% tolerance) to remove the wireless card from inside of the laptop case, and expose the power lines that feed the card. We measured the voltage across these sense resistors by sampling at a frequency of $100KHz$ with an NI USB-6218 DAQ. During the measurements we removed the battery from the computer to avoid interference from charging cycles. We applied a lowpass IIR filter to the voltage, and converted the measured voltage drop and given resistance to current using Ohm's law.[1]

Our final data consists of two time series: one of kernel events, timestamped with the kernel's internal clock, and one of power samples, timestamped with the DAQ's internal clock. We synchronized both time bases in post processing by always surrounding experiments with a small series of ping packets, which showed a very distinctive pattern in the power curve corresponding to a stable sequence of kernel events.

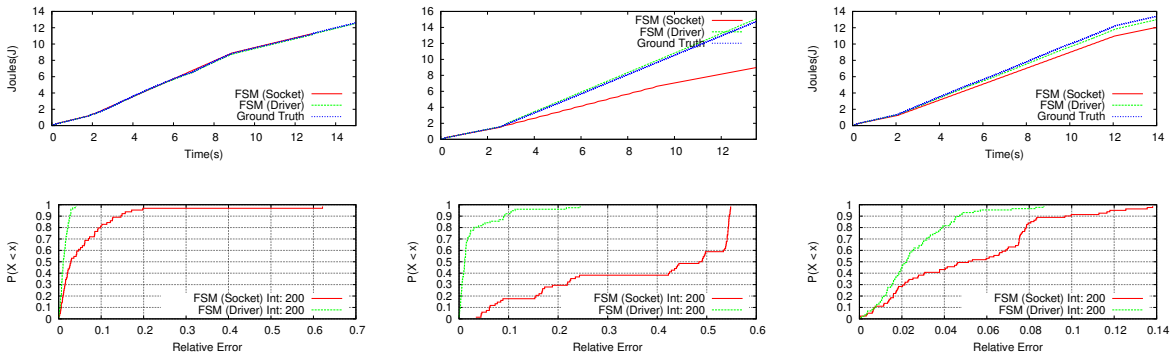
For evaluation of specific experiments, we show the cumulative measured energy curve for the experiment, together with the energy curve predicted by the models. To assess the error, we follow [14] and show the CDF of the relative errors of the energy measured and predicted on successive intervals. For the plots that follow, we compute the errors on intervals of 200ms.

Typical Experiment We ran several microbenchmarks using netperf, sending and receiving UDP datagrams with sizes ranging from 100B to 10KB, with sending intervals ranging from 50 to 500ms. Figure 3(a) shows a typical experiment in which both models performed similarly at the macro level, even though the tail of the per-interval error distribution for the system-call model is significantly longer.

Unsolicited packets

Figure 3(b) shows a case in which the FSM-Sock model performs arbitrarily worse than the FSM-Driver model. In this experiment we start by having the tested node receive UDP packets through netperf (top figure, up to 2.25 s). We then start another process in the remote machine, which sends unsolicited

[1]Note that our method of measuring power draw is not particularly novel or the only way to achieve this. In particular, other works (*e.g.* [14, 10]) have shown that by properly isolating the workload one can obtain the variations in power draw by only measuring the global power draw of the platform.



(a) Periodic receive (200ms, 100B)

(b) Periodic receive (500ms, 100B). Unwanted packets.

(c) Periodic send (200ms, 1000B). Interference with beacons.

Figure 3: Energy consumption and CDF of absolute error percentage between measured and estimated energy for benchmarks using FSMs on the socket and device-driver layers.

UDP datagrams to a port with no listening process.

These unwanted packets cause the radio to be on, and are only dropped at the IP layer. Most importantly, they generate no system call, countering an assumption in [14] that all activities in devices are mediated by system calls. As a result, the FSM-Sock model significantly underestimates the energy consumption, as it assumes that the card is constantly entering the sleep state after each reception. In general, it is not necessarily true that hardware interrupts will generate system calls.

Interaction with beaconing Our second example of a limitation of the system-call-level model is shown in Figure 3(c). In this experiment we had netperf send a 1000B UDP datagram every 200ms. Note how the FSM-Socket model underestimates the energy consumption, and how the median of interval error rates increases from 2 to 5%.

Figure 4 shows why this happens. The actual period that packets are sent in this experiment is 204ms, while the WiFi beacons from the basestation arrive every 100ms. Interestingly, the relative interval between packet transmission and the next beacon change the duration of the apparent tail state after the transmission, which cannot be inferred from the system-call layer alone. As a result, there is no stable value for the tail state, and any single choice of a value in the model would result in an error. We also observed this behavior across other experiments, both with sendind and receiving.

5 Discussion

Limitations of this study The measurements presented here are obviously not comprehensive, neither in terms of workloads nor in terms of hardware diversity. As such, they are intendend as specific examples of when events in the device-driver layer capture power-related events that are not reflected in the system-call layer. However, the events in the lower layer are a superset of the events in the upper layer, and in principle should allow for better and simpler models.

Another point not addressed here is a limitation of power modeling in general [13], in that there may be power-related events that the hardware itself does not expose to the driver, for efficiency, lack of perceived need, or competivite advantage reasons. The device-driver model would fail to capture such behavior as well.

Exposing Power States

Our central argument in this paper is that device

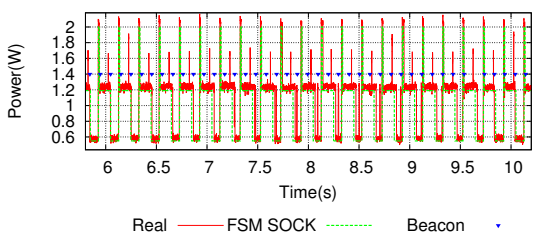


Figure 4: Measured and predicted power curves for periodic send (200ms, 1000B), showing how the actual tail state after each state fluctuates due to the out-of-phase WiFi beacon reception.

drivers should expose power states to the operating system. We are not the first to argue this argument (see, for example, [18]). We simply show an instance in which having such information makes for simple and accurate power models.

The interface to expose power states should be as narrow and uniform as possible, to be implementable by the largest possible number of devices. It could be query-based, such that the OS could implement sampling of power states (uniform sampling in time would work to provide the relative frequency of each power state, at a controllable overhead.) It could also be interrupt-based, such that the OS would have to implement efficient accounting handlers for asynchronous processing. Lastly, device drivers could also expose the time spent on each power state in the last epoch.

The main question here is what the incentives are, and this is still a major challenge to the community. Devices tend to manage their own power, and not to expose too much internal state to their hosts. Past and present research has repeatedly argued, and we echo this, that the OS is in a better position to manage whole system energy [18, 15, 2]. Lastly, Challen and Hempstead [4] recently argued that a platform with a diverse set of hardware components with redundant functionality and different power profiles could form ensembles that would achieve power proportionality. Correct power modeling across all such components would be key for this to work, and could not be done myoptically by each component.

6 Acknowledgments

This research is partially supported by a generous gift from Intel Corporation. The authors would like to thank Ben Greenstein, Ling Huang, Jaideep Chandrashekar, and Prabal Dutta for their insigthful comments and support.

References

[1] SystemTap. `http://sourceware.org/systemtap/`.

[2] F. Bellosa. The case for event-driven energy accounting. Technical Report TR-I4-01-07, June 29 2001.

[3] W. Bircher and L. John. Complete system power estimation: A trickle-down approach based on performance events. In *Proc. IEEE ISPASS*, pages 158 –168, april 2007.

[4] G. Challen and M. Hempstead. The case for power-agile computing. In *Proc. HotOS XIII*. USENIX, May 2011.

[5] T. L. Cignetti, K. Komarov, and C. S. Ellis. Energy estimation tools for the Palm. In *Proc. of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems (MSWiM)*, pages 96–103. ACM Press, August 2000.

[6] G. Contreras and M. Martonosi. Power prediction for Intel XScale processors using performance monitoring unit events. In *Proc. ISLPED'05*, ISLPED '05, pages 221–226, New York, NY, USA, 2005. ACM.

[7] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *Emnets IV*, Cork, Ireland, June 2007.

[8] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. In *Proc. Workshop on Modeling, Benchmarking, and Simulation (MoBS)*, June 2006.

[9] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proc. IEEE WMCSA '99*, pages 2–, Washington, DC, USA, 1999.

[10] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *Proc. OSDI*, pages 323–338, December 2008.

[11] C. Inc. Cc2420 data sheet. `http://www.chipcon.com/files/CC2420_Data_Sheet_1_0.pdf`, 2003.

[12] S. Kellner, M. Pink, D. Meier, and E.-O. Blass. Towards a realistic energy model for wireless sensor networks. In *Proc. WONS*, Garmisch-Partenkirchen, Germany, Jan. 23–25 2008.

[13] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta. Evaluating the effectiveness of model-based power characterization. In *Proceedings of the USENIX ATC*, Portland, Oregon, USA, June 2011.

[14] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proc. EuroSys'11*, pages 153–168, 2011.

[15] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Energy management in mobile devices with the cinder operating system. In *Proc. EuroSys'11*, 2011.

[16] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: A platform for OS-level power management. In *Proc. EuroSys'09*, Nuremberg, Germany, Apr 2009.

[17] T. Stathopoulos, D. Mclntire, and W. Kaiser. The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes. In *Proc. IPSN '08*, pages 383 –394, april 2008.

[18] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Ecosystem: managing energy as a first class operating system resource. In *Proc. ASPLOS-X*, pages 123–132, New York, NY, USA, 2002. ACM.