Efficiently Running Continuous Monitoring Applications on Mobile Devices using Sensor Hubs

Aruna Balasubramanian, Anthony LaMarca*, and David Wetherall
University of Washington, *Intel

ABSTRACT

Smartphone applications that continuously monitor user context are becoming popular for applications ranging from health-care to lifestyle monitoring to participatory sensing. Unfortunately, these emerging apps make poor use of the already scarce energy resources because they continually wake the CPU and other mobile device components for periodic sensing, computation, and communication. We report measurements of three representative Android smartphone apps that show 77% of the battery is wasted by constantly waking up the device components. Moreover, we find that existing power optimization techniques provide only modest benefits for this new class of apps. Using a trace-driven study of the three apps, we estimate that reducing 3G overhead, offloading to the cloud, offloading 3G traffic to WiFi, and using sensor duty cycling, can reduce only 5-10% of the energy costs. For greater energy savings, we make the case for platform support in the form a sensor hub. Sensor hubs are dedicated subsystems that interface with the sensors and radios; they use a micro-controller to support low cost sensing and computation. Our trace-driven analysis shows that sensor hubs can reduce the power consumption of continuous monitoring apps by 61%.

1. INTRODUCTION

Smartphone applications that are always on and continuously monitoring the ambient environment are becoming popular for a diverse range of services. For example, Tasker [10] is a paid Android app with over 250,000 downloads that provides simple context-aware services, such as automatically setting the ringer to silent mode in select locations and during meetings. The popularity of Tasker has spawned similar applications such as Locale [6], and Smart Actions [2]. More ambitious sensing apps continuously monitor user mobility to evaluate Parkinson patients (Ambulation [4]), help users lead a healthier lifestyle by avoiding routes with junk food restaurants (the PEIR project [19]), and encourage users to be more social and active (BeWell [16]). We expect users to run more of this kind of app over time given the tremendous potential of monitoring human behavior and the environment.

Unfortunately, applications that require continuous monitoring are a heavy drain on the mobile's battery. We collect and analyze power traces for three representative applications: a healthcare app [1], a lifestyle monitoring app [4], and a participatory sensing app [7]. We find that only a small fraction (23%) of the energy is consumed for actual computation, networking, and sensing. The remaining energy is wasted in wakeup overhead, either waking up the CPU or the network interface. The result is that running these apps together will drain 90% of the battery in 8.2 hours, even when the phone is not otherwise being used. This power profile is unlike that of traditional foreground apps, such as YouTube and AngryBirds, which consume more power but waste much less energy in overhead for the same total energy consumption.

Our measurements suggest that continuous monitoring apps can run up to four times longer on a single charge if they use energy efficiently. To see which power-saving techniques have the most potential for savings, we conduct a trace-driven study to estimate the benefits of computational offloading [15, 23], offloading cellular data to WiFi [12], and using fast dormancy [18]. These are the techniques in the literature that have shown good power savings with traditional applications. Yet we find that none of these techniques provides more than 10% improvement in power consumption for our three continuous monitoring applications. This is primarily because existing techniques do little to reduce the overhead incurred in waking the CPU.

We conclude that new platform support is needed to run continuous sensing apps in a power-efficient manner. The form of support that we investigate in this paper is a dedicated micro-controller that interfaces with the sensors and the 802.11 radio. This *sensor hub* aims to efficiently support continuous monitoring applications by (1) allowing sensing even when the CPU is asleep, and buffering sensor data for further processing; (2) processing small computational tasks without waking up the CPU; and (3) implementing cheaper WiFi maintenance. Several research and industry efforts have shown the feasibility of

adding sensor processing firmware to micro-controllers and integrating them with smartphones [21, 5].

We extend our trace-driven analysis to estimate the benefits of sensor hubs by micro-benchmarking the real-time performance of a matrix-based sensor processing algorithm on an MSP430 micro-controller as a proxy for a sensor hub. We find that a sensor hub can reduce power consumption of the three representative apps by 61% and reclaim the bulk of the wasted energy. For greater gains that enable continuous monitoring apps to run well, research must overcome two key limitations of sensor hubs: they do not reduce the power consumption of the network interface, nor do they reduce the cost of the sensors themselves. Both factors matter, as real world applications often make use of expensive sensors such as GPS and often communicate with the cloud. While sensor costs will only be improved with the development of more efficient sensors, we believe sensor hubs can be adapted to help reduce 802.11 overhead and further improve the efficiency of continuous monitoring workloads.

2. MEASURED POWER PROFILES

We study the power characteristics of three continuous monitoring applications:

Ambulation [4] is a *health monitoring application* to monitor patients with mobility-affecting chronic diseases such as Parkinsons. The app classifies user's daily movements using data from WiFi, accelerometer, and the GPS.

Acoustic Monitoring [1] is a *lifestyle monitoring application* that periodically samples audio data to make sophisticated inferences about human activity.

Mobiperf [7] is a *participatory sensing application* that allows users to visualize the network performance at a given location. The app periodically measures the 3G network performance for a user and uploads these measurement, along with coarse-grained location information, to a server.

We chose these applications to study because they cover multiple domains and engage all three mobile platform components—sensors, processor, and network. Further, all of the applications periodically upload data to the cloud, either for processing or for visualization.

2.1 Methodology

We conduct all our experiments on Nexus 1 phones that run Android 4.0; the phone is equipped with 802.11 WiFi interface and 3G service provided by T-Mobile. We use the PowerTutor tool [9] to measure power consumption at a 1 second granularity, logging the power of each component: the CPU, the network interface, the sensors, the LCD, and the GPS. To validate the accuracy of this tool, we used a Power Monitor [8] in a tethered environment.

For our experiment, we needed to extend PowerTutor with more fine-grained accounting. PowerTutor provides

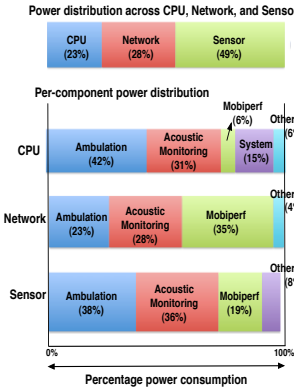

Figure 1: Continuous monitoring experiment.

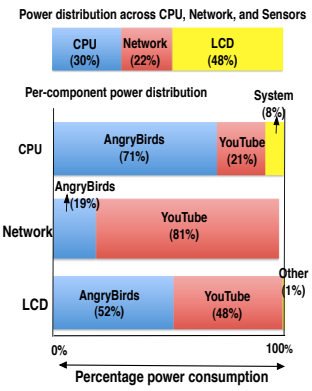

Figure 2: Foreground experiment.

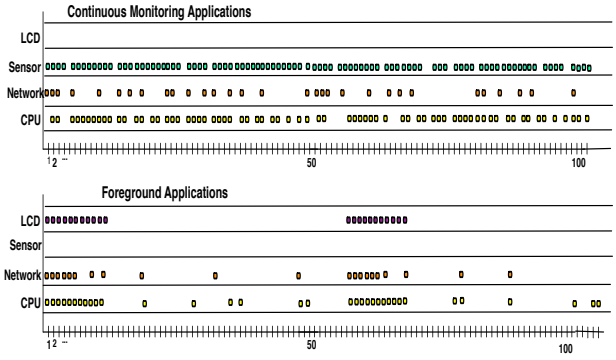

Figure 3: Comparing the active times of the continuous monitoring vs the foreground experiment. The foreground application is active for 2 1-hour sessions.

the CPU power of each application, including the system power consumption. For the network, we log the amount of data sent/received by each app at 1-second granularity, and charge the applications an equal fraction of the network cost for that second. WiFi and 3G networks also incur a set-up and tear-down cost, consuming a significant portion of the network power [13]. We use the application logs to proportionally allocate the set-up/tear-down cost to the applications.

Accounting for the sensor power consumption is more difficult. Sensor measurements require the CPU to be active. Even if the sensor measurement takes only a fraction of a second, waking up the CPU consumes considerably more power due to wake-locks [20]. We assign this CPU power consumption to the application that requested the sensor reading, but only if the CPU was idle before the sensor reading.

2.2 Workloads

Continuous Monitoring Experiment: We run Ambulation, Acoustic Monitoring, and the Mobiperf app simul-

taneously on the phone. This workload mimics a future with popular continuous monitoring apps, and allows opportunities to save energy with existing techniques by coordinating app activity. In this experiment one of the authors carries the phone with them for 3 regular work days and the PowerTutor application logs the power trace. The phone's WiFi is authenticated to connect to both the users home and work WiFi. Each trace is logged until 90% of the battery is discharged. The average trace length is 8.2 hours. The phone uses WiFi by default, and the 3G network when WiFi is not available.

Foreground experiment: We measure the power consumption of two popular foreground apps: YouTube and AngryBirds. We run the foreground application one after the other for 30 minutes each for two sessions, and the phone remains idle for the rest of the experiment. As before, the phone is carried through regular work days and has access to work and home WiFi. Also as before, we collect power traces until 90% of the battery is discharged. The average trace length is 8.4 hours.

2.3 Power Profile Results

Figure 1 shows the power profile of the continuous monitoring experiment. Nearly half of the power (49%) is spent on sensing. The CPU only consumes 22% of the total energy due to the low computational needs of the three applications. The apps use all three components (sensors, network, and CPU), but to varying degrees.

Figure 2 shows the energy distribution of the foreground experiment. Over 50% of the energy is spent on display, and no energy is spent on sensing. While YouTube is network centric with little CPU consumption, AngryBirds is CPU-intensive with little networking. Note that the total energy spent by both experiments are about the same (both run until 90% of the battery is discharged). In summary, *while the display is the most expensive resource for foreground applications, for the continuous monitoring applications sensing uses the most power.*

To dig deeper into the differences between foreground and continuous monitoring applications, we plot the active times of the device for one example day. We divide the trace into 5 minute slots. We consider the device to be active during a slot if it draws more than the base power for at least 1 second in the 5 minute period.

Figure 3 shows the active times for both the continuous monitoring and the foreground experiment. Although the total power consumption is the same, the power profiles look markedly different. In the continuous monitoring experiment, the sensor subsystem is active almost throughout the trace and the CPU is active for large portions of the trace. In the foreground experiment, the components are active only when the foreground application is active (2 sessions, 1 hour each); the device is idle for large portions of the trace. Varying the length of the time slots resulted in a quantitatively similar results (now shown here).

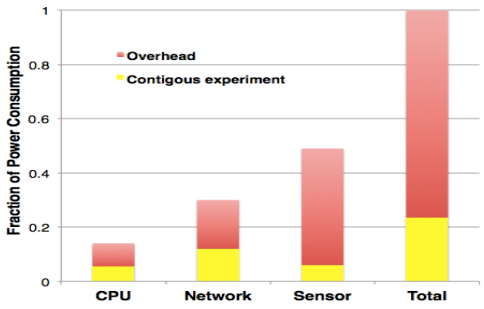

Figure 4: Quantifying the energy overhead. If the application performs all its tasks contiguously, 77% of the power can be saved.

In summary, *continuous monitoring applications require the device to be active almost all of the time* even though they are only intend to run for a small fraction of the total time.

2.4 Estimating Wasted Power

Applications that frequently wake the CPU or the network interface incur a large power overhead that represents wasted energy. To estimate this overhead, we use our trace data to find the power that would be consumed if all the application tasks were performed contiguously. This execution is clearly not feasible as a real power-saving strategy, since the apps need the sensor data to be collected at regular intervals and not all at once. However, it is informative for our goal of bounding overhead. In effect, it gives the absolute minimum power required to run each application as-is but with no wakeup overhead.

For the network, we measure the power consumed if all application data were sent as one block. We exchange the data over TCP to/from a known server, and repeat the experiment for WiFi and 3G traffic. For the sensors, we measure the power consumed to obtain sensor readings continuously, the same number of times as the original experiment (§2.2).

For computation, we identify the core computational unit in the Acoustic Monitoring and the Ambulation application. Acoustic monitoring performs feature selection on the audio data, and the Ambulation app determines if the GPS is needed for localization based on the accelerometer reading and WiFi signature. We measure the power consumption for running this core computation continuously, the same number of times as the original experiment. We ignore Mobiperf because it performs very little computation.

We find (Figure 4) that the three applications need only consume 23% of the expended power; i.e., 77% of the power is wasted on overheads. This implies that the device could run almost 4 times longer if the applications

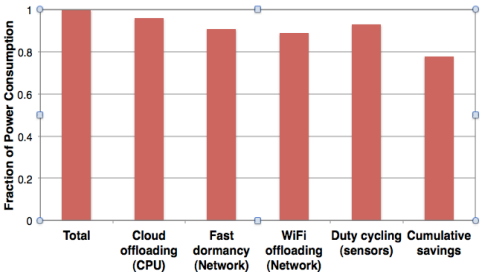

Figure 5: Evaluating existing power optimization techniques for continuous monitoring workloads.

ran as efficiently as possible. A large fraction of this overhead is incurred by the sensing task; the three applications perform frequent sensing, which requires the CPU to be woken up often.

In contrast, the same experiment shows far less overhead for the foreground workload. For the foreground apps, network power consumption reduces by 7% when all application data are sent contiguously. This modest gain is because foreground applications have relatively few short data transfers. The display power consumption remains unchanged. Our analysis shows that the CPU is in the high frequency mode continuously for 90% of the active time, indicating that there is little overhead; we are unable to more accurately measure the power for contiguous computation without the source code.

3. A CASE FOR SENSOR HUBS

3.1 Existing power-saving techniques

Given its importance, researchers have proposed several techniques to reduce smartphone power consumption. They include: (1) cloud offloading [15, 23]; (2) offloading cellular traffic to WiFi [12]; (3) reducing the 3G tail [12, 18, 22]; and (4) using low-power sensors to reduce the use of high-power sensors [14]. We ignore (4), since the only app using a high-cost sensor, Ambulation, already implements low cost location tracking by using acceleremeters and WiFi scans to minimize GPS usage.

We evaluate the best case power benefits when using each of these techniques:

Computational Offload: our apps offload all computation to the cloud. We assume that the applications exchange only 10% of the sensor data with the cloud to perform the computation.

Fast dormancy: we assume that the 3G tail energy is completely eliminated.

WiFi offload: we let all of the data sent over the 3G interface be sent over the more efficient WiFi interface. We assume that WiFi is always available.

Sensor duty cycling: we coordinate the wake up schedules of the applications. The CPU still needs to wake up at least once a minute.

Figure 5 shows the result of our trace-driven evaluation. Disappointingly, most techniques provide less than 10% improvement in power consumption. Even if the power reductions add up, the cumulative savings is at most 22% (with WiFi offloading superseding fast dormancy). The main reason that the benefit is modest is that the techniques do little to reduce the CPU wakeup overhead. Instead, we argue that a better solution to the power problem for continuous monitoring apps is a low-power co-processor platform that will allow the main CPU to be idle for long periods of time. We call this platform a sensor hub and describe it next.

3.2 Sensor hubs

A sensor hub is a dedicated subsystem that interfaces with the sensors and radios, and serves as an intermediary between the main platform and the sensors. Sensor hubs are ideally suited for reducing wakeup overhead because they allow sensing without waking the host CPU. Our experiments on continuous sensing suggest that large power savings can be had if applications perform sensor measurements less frequently. However, application functionality can be severely affected if sensor data is not collected at the requisite frequency. Sensor hubs resolve this tension by collecting and buffering sensor data that applications fetch later when needed.

Moreover, sensor hubs can perform simple processing to further reduce the need to wake up the CPU. For example, many sensing applications [16, 1] compute simple features from the sensor data such as min, max, or average over a sliding window. These computations can be executed even on the simplest of micro-controllers; the sensor hub can then wake up the CPU only under certain thresholding conditions.

Figure 6 shows an example sensor hub architecture. Sensor hub designs typically use of a micro-controller in lieu of a microprocessor, because of its low cost and power efficiency. For example, the MSP430 micro-controller running at a maximum speed (1MHZ) draws 0.3mA of current, an order of magnitude less than smartphones. The micro-controller buses (12C and SPY) draw negligible power and the MSP430 costs less than 25 cents. Recent research has shown the feasibility of interfacing smartphones with a MSP430 processor to perform sensor measurements [21]. In addition, a number of commercial sensor hubs are also available such as the Lapis ML610Q792 [5] that are packaged for smartphones.

While using sensor hubs to offload CPU tasks is straightforward, we believe that sensor hubs can also be used to reduce the network overhead. Connection maintenance on WiFi networks is expensive [13], as the 802.11 connection requires that the NIC wake up frequently to check if the access point has any packets buffered for it. Typically,

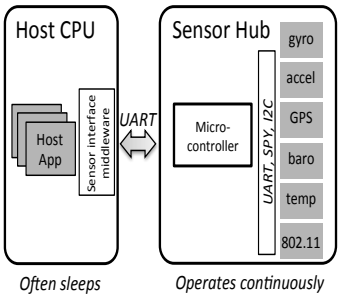

Figure 6: The Sensor Hub Architecture.

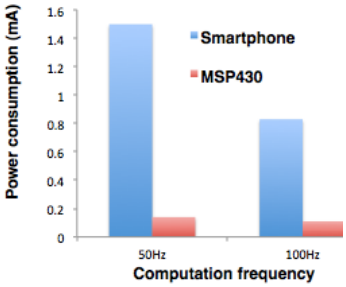

Figure 7: Compute power to run a matrix-based algorithm on a micro-controller.

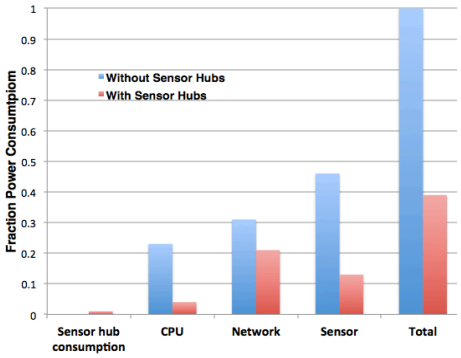

Figure 8: Evaluating the power consumption benefits of using a sensor hub.

the NIC checks for buffered packets every 100ms. By delegating the management of the 802.11 connection to an 802.11-capable sensor hub (such as the CC3000 Wi-Fi MSP430), the maintenance can be performed at a much lower power cost.

3.3 Benefits of sensor hubs

To estimate the power savings offered by a sensor hub, we start by executing a representative sensor processing algorithm on the popular MSP430 micro-controller (TI part MSP430F5172). From a software-based IMU implementation, we use a matrix-based algorithm that updates pose and heading estimates [3] using accelerometer and gyroscope data. We ran the micro-controller at .5 and 1 Mhz which allowed the algorithm to process 50 and 100 sensor updates per second, respectively. Figure 7 shows that the micro-controller draws only 0.15mA processing 50 sensor updates/second and 0.3mA at 100 updates/second.

By combining this data with our application traces, we can estimate the benefits of buffering sensor readings, offloading computation, and reducing WiFi maintenance. The trace data notes the times when the application obtained sensor readings, when it performed computation, and when it sent/received data. We assume that the applications offload both computation and sensor measurements to the micro-controller, but wake up for each network event. Further, we assume an 802.11 chipset that allows the microcontroller to take over WiFi connection maintainance when the main CPU is asleep.

Note that the radio and the sensors themselves still consume the same amount of power as previously; the sensor hub only reduces the CPU wakeup cost. Finally, we assume that the sensor hub runs at maximum speed (1 Mhz) and, as in our measurements, draws 0.3mA.

Figure 8 shows the percentage power savings if a sensor hub were used for the continuous monitoring experiment (§2.2). Overall, a sensor hub reduces the power consumption by 61%. It saves nearly 84% of the CPU cost, and reduces the sensor cost by 68%. Note that the sensor hub does not reduce the cost of the sensors themselves, so expensive sensors such as GPS still draw considerable power.

The reduction in network power consumption is much more modest. The WiFi maintenance itself is only a small part of the total network cost, and as a result sensor hubs do not reduce the network cost substantially. Part of our future research agenda is to explore WiFi data offload to the sensor hubs themselves. It is unclear if in doing so the communication performance will be sufficient for continuous monitoring applications, and if meaningful power reduction can be obtained.

4. RELATED WORK

The approach of using a tiered system for energy efficiency has been used in several settings. For increasing the idle time of expensive NICs, Somniliquy [11] relies on a secondary embedded controller. Similarly, Wake on wireless [24] uses a lower energy Bluetooth communication link to wake up the main network interface. Sorber *et. al* [25] use a tiered system, with a sensor node embedded in a PDA embedded in a laptop [25].

More recently, Priyantha *et. al* [21] designed a sensor co-processor for smartphones, and report 95%+ reduction of power consumption for a simple pedometer application. Their system, along with commercial variants [5], shows the ability of a sensor hub to reduce the overhead of background sensing workloads. Our work expands on Priyantha *et. al* in two ways. First, we quantify the overhead for this class of applications and show that existing power optimization techniques only provide modest benefits. Second, we do so by measuring a set of three representative, real-world apps that include networking and expensive sensors such as gyroscope and GPS, neither of which is easily optimized with a sensor hub. Our

analysis suggests a useful but lower (61%) estimate of the power savings offered by a sensor hub that is perhaps more realistic.

An alternative to a sensor hub design is to embed buffering and inference support into the sensor themselves. Such an approach is referred to as "smart sensing" and commercial versions exist (e.g.: The Micronas Hall-effect sensors) This approach work well for special-purpose devices, but scales poorly and suffers from not being as programmable as a sensor hub.

Finally, Lin *et. al* [17] propose an automatic and transparent way of distributing an application across the main host and an embedded co-processor such as a sensor hub. These and other techniques will be necessary to make sensor hubs easy to program.

5. CONCLUSIONS

In this paper we characterize the power profile of a new class of applications that require continuous monitoring of the environment. Using Android measurements, we explore three representative continuous monitoring applications that drain 90% of the phone battery within 8 hours, even when the phone is not actively used. A trace-driven analysis showed that these applications only spend a small fraction (23%) of their power doing useful sensing, computing, and networking. The other 77% is overhead. Using a trace-driven study we evaluated a number of power-saving techniques including cloud offload, fast dormancy, WiFi offload, and a dedicated sensor hub. Only the sensor hub provided substantial improvements in efficiency (61%), with the others yielding 5-10% gains for our measured workloads. However, the sensor hubs were not able to significantly reduce the network or GPS cost. We identified the idea of adding an always-on 802.11 networking capability to a sensor hub for improved platform support as a research challenge.

6. REFERENCES

[1] Acoustic monitoring : http://research.cens.ucla.edu/.

[2] How to best use the motorola smart action app : http://www.gottabemobile.com/2012/02/07/how-to-best-use-the-motorola-smart-actions-app/.

[3] Inertial measurement unit : http://en.wikipedia.org/wiki/Inertial_measurement_unit.

[4] Inferring everyday mobility states using mobile phones: http://research.cens.ucla.edu/urban/2011/part01.pdf.

[5] Lapis ml610q792 : http://www.lapis-semi.com/en/semicon/miconlp/ml610q792.html.

[6] Locale for android : http://www.twofortyfouram.com/.

[7] Mobiperf: http://mobiperf.com/.

[8] Monsoon power monitor : http://www.msoon.com/.

[9] Powertutor : http://powertutor.org/.

[10] Tasker for android : http://tasker.dinglisch.net/.

[11] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta. Somniloquy: augmenting network interfaces to reduce pc energy usage. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, NSDI'09, pages 365–380, Berkeley, CA, USA, 2009. USENIX Association.

[12] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3g using wifi. In *MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 209–222, New York, NY, USA, 2010. ACM.

[13] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Proc. ACM IMC*, November 2009.

[14] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, and L. P. Cox. Enloc: Energy-efficient localization for mobile phones. In *INFOCOM'09*, pages 2716–2720, 2009.

[15] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62, New York, NY, USA, 2010. ACM.

[16] N. D. Lane, T. Choudhury, A. Campbell, M. Mohammod, M. Lin, X. Yang, A. Doryab, H. Lu, S. Ali, and E. Berke. BeWell: A Smartphone Application to Monitor, Model and Promote Wellbeing. In *Pervasive Health*, May 2011.

[17] F. X. Lin, Z. Wang, R. LiKamWa, and L. Zhong. Reflex: using low-power processors in smartphones without knowing them. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '12, pages 13–24, New York, NY, USA, 2012. ACM.

[18] H. Liu, Y. Zhang, and Y. Zhou. Tailtheft: Leveraging the wasted time for saving energy in cellular communications. In *MobiArch 2011*. ACM, 2011.

[19] M. Min, R. Sasank, S. Katie, Y. Nathan, B. Jeff, E. Deborah, H. Mark, H. Eric, W. Ruth, and B. Péter. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, MobiSys '09, pages 55–68, New York, NY, USA, 2009. ACM.

[20] 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, New York, NY, USA, 2011. ACM.

[21] B. Priyantha, D. Lymberopoulos, and J. Liu. Littlerock: Enabling energy-efficient continuous sensing on mobile phones. In *IEEE Pervasive*, 2011.

[22] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Top: Tail optimization protocol for cellular radio resource allocation. In *Proceedings of the The 18th IEEE International Conference on Network Protocols*, ICNP '10, pages 285–294, Washington, DC, USA, 2010. IEEE Computer Society.

[23] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan. Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 43–56, New York, NY, USA, 2011. ACM.

[24] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, MobiCom '02, pages 160–171, New York, NY, USA, 2002. ACM.

[25] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: hierarchical power management for mobile devices. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, MobiSys '05, pages 261–274, New York, NY, USA, 2005. ACM.