EERS: Energy Efficient Responsive Sleeping on Mobile Phones

Bodhi Priyantha, Dimitrios Lymberopoulos, Jie Liu
Microsoft Research
One Microsoft Way, Redmond, WA 98052
{bodhip, dlymper, liuj}@microsoft.com

ABSTRACT

We present the concept of *Responsive Sleeping*, where the mobile phone can continue to sense its environment in the sleep mode. This capability enables novel applications in user interaction, context awareness, and people monitoring. However, the state of the art phone architectures today require the application processor (AP) to be active to read sensor data. The energy consumption makes responsive sleep impractical. In this paper, after analyzing the root cause of mobile sensing energy consumption, we present a new mobile phone architecture that uses heterogeneous multi-processors (or a multi-core processor) to achieve energy efficient responsive sleeping (EERS). We present a prototype called LittleRock and evaluate its energy benefit for responsive sleep through continuous sensing examples.

1. INTRODUCTION

Mobile phones today follow their users in almost every activity they engage in. The ubiquity, mobility, and connectivity of smart phones have made them ideal platforms for personalized mobile services, evident from the large number of applications available for various mobile platforms. Apart from having a reasonably powerful processor and graphics capability, current high-end smart phones have a rich set of built-in sensors, such as GPS, microphone, camera, accelerometer, ambient light, compass, gyro, and pressure sensors that enable measuring various phenomena on and around the phones and thus their owners. Location-aware services, natural user interfaces, and many games rely on these sensors to provide a superior user experience.

While it is most common for mobile applications to use the sensors *on-demand*, where sensor reading operations are initiated by the foreground process, application developers have also shown the value of performing continuous sensing in the background, even when the phone is not receiving direct user attention. For example, in *UbiFit* [3], Klasnja1 et. al. studies user behavior changes when they are fed with fitness information such as exercise tracked by wearable accelerometer and barometer sensors. In [6], a single wearable sensor is used to estimate the calori expenditure of a user. In *Playful Bottle* [2], Chiu et. al. attached a mobile phone to a water drinking device and used images and motion to detect the frequency and quantity of the user's water consumption. In SoundSense [8], Lu et. al. designed a mobile phone application that uses sound signatures to detect user location and activity. In the Mobile Millennium project [1], mobile phones were used as tripwire sensing devices to collectively detect traffic conditions.

All of these applications require the phone to continuously sample its sensors independently of whether the user is interacting with the phone. We call the capability that the phone can continuously sense while appears to be sleeping *Responsive Sleeping* (RS). In a responsive sleeping mode, the phone does not draw user attention or require a foreground application. However, in the background, it can continuously monitor sensor input and wake up the phone if interesting events happen.

Responsive sleeping can also greatly improve user experience with the phone, as if the phone is always on. For example, if a phone can detect that it is being picked up from the table and is approaching the face, it can automatically enter the voice command mode. If the phone can continuously monitor audio input, it can run speaker recognition to detect the user's company; then it can use this information to pop up reminders or set UI preference (e.g. adjusting ring tone or vibration). Another example is "geo-fencing", where an application registers a geographical region of interest, so that an event is triggered to wake up the phone and activate the application when the phone enters that region.

Responsive sleeping can be based either on sensors on the phones, or on external sensors that communicate wirelessly with the phone. In both cases, the energy consumption for supporting RS is challenging. For example, we will show in section 2 that a simple pedometer application can drain a smart phone battery in a few hours. This energy inefficiency is not due to the system software overhead, e.g., task scheduling, but rather due to the fundamental limitations of current phone architecture. That is, to acquire any sensor data or communicate with detached sensors, the application processor (AP) must be active and running. Active APs typically consume hundreds of milli-Watts (mW) of power. In section 2, we also show that sensor duty cycling [10] does not solve the energy inefficiency problem, since it takes up to a second for the AP to wake up from the sleep mode and restore the state for taking sensor readings.

Enabling *energy efficient responsive sleeping* (EERS) motivates us to rethink the mobile phone sensing architecture. In this paper, we propose a new mobile phone sensing architecture that can support EERS by breaking the tight coupling between sensors and the AP. With heterogeneous multi-processors or multi-core processors, a low power microcontroller or processor core can manage the sensors without shortening the battery life noticeably. As a prototype platform, we designed LittleRock, which adds a small, energy efficient co-processor to the phone to offload sensing tasks to this small processor. All the available sensors and

Sensor	Sensor State			HTC Touch Pro State (mW)		
	Active(mW)	Sleep(μW)	1Hz Sampling(mW)	Active (1680)	Idle (399)	Sleep (7.56)
Accelerometer	0.6	3	0.018	0.001%	0.004%	0.23%
Temperature	0.225	3	0.02	0.001%	0.005%	0.26%
Pressure	1.8	0.3	0.02	0.001%	0.005%	0.26%
Compass	2.7	7.5	0.5	0.03%	0.125%	6.61%
Gyro	19.5	15	1	0.06%	0.251%	13.22%
GPS chip (1MIPS CPU required)	214 (acq. state)	5	30	1.78%	7.518%	397%
Total (with GPS)	238.825	34.1	31.558	1.88%	7.91%	417%
Total (without GPS)	24.825	29.1	1.558	0.093%	0.39%	20.6%

Table 1: Power consumption of different types of sensors and their overhead (assuming 1Hz sampling rate) on the overall power consumption of an HTC Touch Pro phone in 3 representative power states.

short range radios on the phone are connected to the small processor, enabling the rest of the phone to go into the sleep mode. While the phone is sleeping, the co-processor can continue to acquire samples, process sensor data, and communicate with external sensors, all at a low energy overhead. Since the two processors are tightly integrated, data between them can be easily buffered and quickly exchanged on demand. We discuss EERS architectures in section 3 and the design of LittleRock in section 4.

Multiple sensing modalities can usually achieve the same goals with different energy and processing requirements. The addition of the sensing processor complicates this problem by introducing additional trade offs between energy and computation. In addition, the heterogeneous multiprocessor architecture brings complexities to software design and application programming. We discuss key hardware and software challenges toward responsive sleeping in section 6.

2. MOBILE ENERGY BREAKDOWN

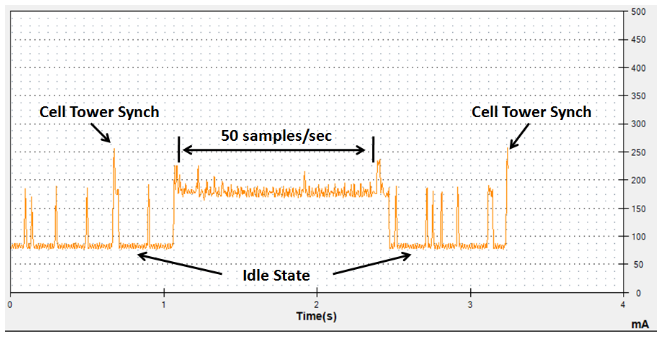
Battery life is one of the most critical design parameters for a phone. Every new feature introduced, either it is hardware or software, has to minimize its impact on the battery life. Consequently, although continuous sampling and processing of sensor data enables new application modalities, it is necessary that these additional features do not severely reduce the phone battery life.

Table 1 shows the overhead introduced by popular types of sensors in the power consumption of a mobile phone, the HTC Touch Pro running Windows Mobile 6.1. The power overhead for each sensor is expressed as a percentage of the power consumed by the HTC phone in 3 representative power states: *Active*(1680mW), *Idle*(399mW), and *Sleep*(7.56mW). In the *Active* state, the phone is exercising its CPU by running random computations while simultaneously downloading data over the 3G radio. In the *Idle* state the phone is turned on, but there is no load imposed on the CPU beyond the background services introduced by the operating system. Also, no data is being sent or received over the 3G radio. In the *Sleep* state the main processor is in sleep mode.

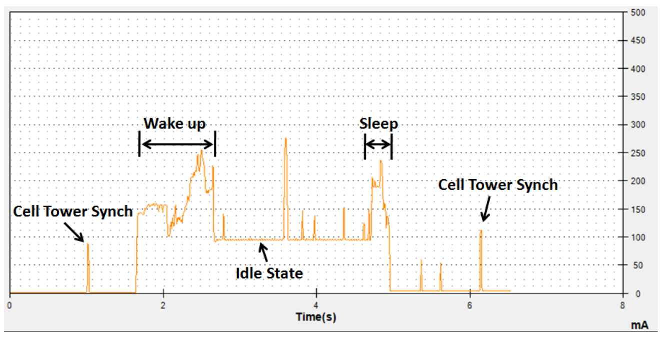
When all the sensors listed in Table 1 are powered up, the overall power consumption of the phone at the *Active*, *Idle* and *Sleep* states increases by approximately 1.88%, 7.91%, and 417% respectively. Note that in all cases, the GPS sensor is responsible for 95% of the overall power overhead. However, as recent work has demonstrated, more energy efficient location sensing can be achieved by properly combining cell tower triangulation and wifi fingerprinting to enforce more aggressive duty cycling of the GPS sensor [7],[9],[12]. Without the GPS sensor, the total sensor power overhead at the *Active*, *Idle* and *Sleep* states becomes 0.093%, 0.39%, and 20.6%.

Even though the continuous operation of the hardware sensors comes at a low power overhead, the process of accessing and processing sensor data on current state-of-the-art phones is extremely expensive. The reason is that for every sensor sample acquired by the phone, the main processor and associated components have to be active, creating a large energy overhead. To better illustrate the impact of continuous sensing on the battery life of current phones, consider an example application where the accelerometer on the phone is continuously sampled at a fixed frequency to perform a variety of tasks such as user activity recognition, dead reckoning-based indoor navigation, and step counting (pedometer) [5, 11, 4]. Figure 1(a) shows the power consumption of an HTC Touch pro phone while sampling the built-in accelerometer at the rate of 50 samples per second. When sampling the accelerometer, the overall power consumption of the phone jumps to approximately 756mW compared to the 7.56mW and 399mW of power consumption of the phone in the *Sleep* and *Idle* states respectively. This increase in power consumption is due to the fact that the CPU of the phone has to be active in order to acquire and store each accelerometer sample. In practice, this means that when sampling the sensors, the phone has to consume approximately 756mW, which is two orders of magnitude higher than the power consumed by the phone in the *Sleep* state.

Besides increasing the power consumption due to sampling, continuous sensing introduces another major bottleneck by essentially preventing the phone from moving to its *Sleep* state. The reason can be clearly seen in Figure 1(b), which shows the power trace for waking up and putting a phone into sleep. The phone needs approximately 900ms to move to and 270ms to exit from the *Sleep* state. As a result, a full transition between the phone's *Sleep* and *Idle* states takes more than a second. Because of this overhead, even when continuous sampling is required at a very low sampling rate, such as 2 samples per second, the phone does not have enough time to transition to and recover from the *Sleep* state and still acquire the next sensor sample on time. As a result, in order to meet the timing requirements for continu-



(a)



(b)

Figure 1: Current drawn from the HTC Touch Pro while (a) sampling the accelerometer at a rate of 50 samples per second and (b) performing a full sleep cycle.

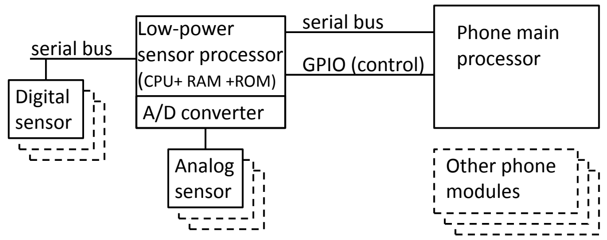


Figure 2: EERS phone architecture block diagram

ous sensing, the phone has to be *Active*, consuming approximately 756mW. Given the 1340mAh battery capacity, an HTC Touch Pro that continuously samples its accelerometer would last for ≃6.7 hours, ignoring other services such as phone calls, SMS send/receive, and 3G data traffic. With these other services, the battery life will reduce even further, making continuous sampling and processing of sensor data impractical on current phones.

The goal of EERS is to enable continuous sampling and processing of sensor data without significantly impacting phone's battery life.

3. SYSTEM ARCHITECTURE

To achieve EERS, continuous sampling and, to an extent, processing sensor data must be decoupled from the application processor of the phone. To achieve this, one can introduce a low power microcontroller, or a low power core in the multi-core processor to manage the sensors. This low power microcontroller or the processor core enables most parts of the phone to enter a low power *sleep* state, while the low-power sensor processor is continuously sampling and processing sensor data at a low-power overhead.

Figure 2 shows the block diagram of the proposed sensing architecture. In this architecture, various low-power analog and digital sensors are attached to a sensor processor. The sensor processor is typically a low-power microcontroller that consists of a CPU, RAM, ROM, and various peripherals such as serial communication buses. The sensor processor is interfaced to the main phone processor using a serial bus and multiple control signals.

3.1 EERS Architecture Features

We highlight the following benefits of introducing a low power sensing processor in EERS architecture:

Low power operation. Since a low-power processor with a power consumption similar to that of a typical sensor is used as the sensor processor, waiting during sensor readings and control does not impose high energy overhead. For example, the MSP430 family of low-power processors consumes ≃ 1mW of power when operating at 1MHz.

Due to the simpler hardware architecture, a low-power processor can transition between sleep and active modes within a very short time. For example, the MSP430 class of processors can switch between the sleep (≃ 0.01mW) and active (16MHz clock, ≃ 20mW) states in $<5\mu$s. This short transition time enables the sensor processor to be heavily duty cycled to reduce the average power.

Programmable context-aware decision making. EERS provides a programmable approach to deciding when to wakeup the main processor based on sensor data. This decision making can be customized to meet the demands of the current user and the set of applications running on the phone. Since the sensor processor has access to nearly simultaneous readings from multiple sensors, the proposed architecture enables richer context aware decision making through sensor fusion.

Real time sensing and event handling. A typical microcontroller has multiple hardware modules such as counters, timers, A/D converters, and serial buses that can operate simultaneously. This hardware parallelism and the typical multi MHz processor clock speeds make it possible to achieve almost real time response when sampling and processing sensor data (assuming a light-weight processing workload).

3.2 Implementation Options

There are three options for introducing the proposed sensing architecture in mobile phones.

Use an existing microcontroller. Phones already have dedicated microcontrollers implementing specific functions. One good example is the capacitive touch controller which interprets various user touch events and communicates to the main processor using a serial bus. One possible implementation to enable EERS is to enhance one of these processors and attach the sensors to this processor. This can be a relatively low cost implementation option, since this requires only a simple modification to existing phone platform architecture.

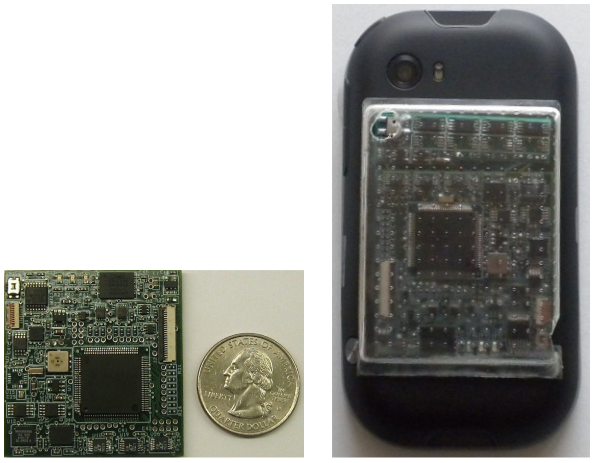

(a) (b)

Figure 3: (a) The *LittleRock* prototype (b) The *LittleRock* prototype attached to the back of a phone.

Add a microcontroller. In this option, a dedicated low-power microcontroller is added to the phone. This option increases the total number of components on the phone. This is likely to be the most costly option out of the 3 mentioned here. However, given most low-power processors are relatively inexpensive (<$1) in typical cellphone volumes, the new application modalities enabled by EERS are likely to offset this additional cost.

Add a low-power core. This option adds a low-power core to the phone's main processor. This is likely to be the least cost option, since incremental cost of adding a simple, low-power core on to the phone's main processor is likely to be very small. However, this option is not likely to be the first option adopted by phone manufacturers, since this requires changes to the phone's main processor itself.

4. PROTOTYPE

We prototyped and evaluated the EERS sensing architecture using LittleRock, a prototype hardware platform consisting of multiple sensors and a low-power processor that interfaces directly to the main processor of a prototype phone. The main goals of this prototype were to experiment with various context-aware applications that can benefit from EERS, and to evaluate the energy overhead due to the proposed sensing architecture.

Our prototype connects to the phone using a wired interface. The resulting platform has a form factor similar to that of a typical mobile phone, enabling us to conduct various user studies on various EERS-enabled applications. This section describes the details of LittleRock prototype.

LittleRock consists of four functional modules: the processor, digital sensor, analog sensor, and the phone interface.

The processor module consists of an MSP430F5438 processor with 16kB RAM and 256kB flash memory. The processor can be clocked up to 18MHz. This processor also has a large number of parallel and serial inputs and outputs, making it possible to attach additional sensors than those already built in to our prototype. This prototype also has a smaller MSP430 processor (MSP430F2013) for on demand reprogramming of the MSP430F5438 processor. The processor module also contains an 8MB flash storage.

Batch size	Power		
	Phone	LittleRock	Hybrid
10	693	0.21	702
50	693	0.21	171
100	693	0.21	90
500	693	0.20	25

Table 2: Average power required to acquire and process 500 samples at a 10Hz sampling rate at different processing batch sizes

The digital sensor module contains a temperature sensor, a 3-axis accelerometer, a barometer, and a 3-axis compass module connected to the main processor through an I2C bus.

The analog sensor module consists of sensors that have analog outputs. In particular, this module contains an X-Y axis gyroscope and a Z axis gyroscope that collectively provide 3-axis gyroscopic data. To reduce the impact due to processor generated digital noise, and to provide better resolution than what is possible with the built in A/D converter of the processor, we used 3 external 16 bit A/D converters to digitize the gyroscope outputs.

The phone interface consists of a wired connection between LittleRock and the expansion connector on a prototype phone. LittleRock and the phone communicate over a 4 wire SPI bus. LittleRock is directly powered from the phone's battery, through the expansion connector. The phone's IO voltage exposed by this connector indicates if the phone is powered on or off. A GPIO pin enables LittleRock to interrupt and wake up the main processor.

5. EVALUATION

In this section we evaluate the performance of a pedometer application that counts user steps based on periodic accelerometer samples. A pedometer is a classic application that benefits from EERS, where the accelerometer has to be continuously sampled even when the phone is in *sleep* mode.

The pedometer application samples a 3-axis accelerometer at 10 Hz, after collecting a batch of n samples, it examines the magnitude variation of acceleration to detect user step events. We evaluate the energy consumption of a pedometer application under three different configurations: running on the phone, running on LittleRock, and running on a hybrid of phone and LittleRock. In the hybrid approach LittleRock buffers the batch of n samples and sends them to the phone for updating the step count.

Table 2 shows the average power consumption of the three different hardware configurations. The column under "Phone" is the most energy inefficient configuration since it consumes $\simeq$ 700mW. Note that this number does not change with the processing batch size, since at the 10Hz sampling rate, the phone continues to be in the *active* state due to the large sleep transition time.

The LittleRock only configuration consumes $\simeq$ 0.2mW. This corresponds to more than 3 orders of magnitude improvement in the power consumption compared to the phone only configuration. Unlike the phone, the low-power processor on LittleRock can transition to sleep mode almost instantly. This, combined with the lower power consumption of the processor, reduces the overall power overhead in LittleRock configuration. Note that, in this configuration, the power consumption reduces slightly with larger batch sizes due to the amortization of fixed processing overhead over a larger batch of data.

Table 2 also shows that the hybrid approach can be significantly energy efficient for large batch sizes. This is because larger batch sizes enables the phone to spend more time in the sleep mode, while LittleRock is sampling and buffering sensor data.

6. RESEARCH CHALLENGES

The proposed architecture for enabling EERS fundamentally changes how applications interact with sensors. This brings additional systems challenges.

Sensor processor selection. In the proposed architecture, sampling and processing of sensor data is offloaded to a sensor processor or a sensor processor core. However, determining how much processing capability and functionality the sensor processor should possess is a challenge.

If the sensor processor does not have enough processing capability, the main processor has to be woken up more often, resulting in high energy consumption. On the other hand, too much functionality in the sensor processor increases the processor sleep currents and wakeup times, and thus the average power. Consequently, the processor selection requires a careful evaluation of the anticipated resource requirements.

Sensor processor resource management. The sensor processor samples and processes sensor data on behalf of multiple user applications. This can result in multiple user applications competing for resources on the sensor processor. Such competing applications raise two challenges.

The first challenge is ensuring fair sharing of sensor processor resources among multiple applications. The second challenge is preventing applications from overloading the sensor processor, which can lead to unpredictable behaviors due to effects such as stack overflows. To address these concerns, the sensor processor needs to employ strict resource counting and management of its memory, processing, and energy resources.

Sensor API. The user applications access the sensor sampling and processing services of the sensor processor through a sensor API. This API should be flexible enough to access multiple services provided by a generic programmable processor, while being simple enough for application developers to use this API without spending too much effort.

Application partitioning. Under the proposed architecture, an application that uses sensor data spans across the sensor processor and the main processor. With this, an application developer needs to decide how to partition an application across these processors. Offloading too little to the sensor processor results in spending too much energy on the main processor, while offloading too much can overburden the sensor processor. Enabling application developers to make the best decision on how to partition an application will require support for fine-grained application profiling.

7. CONCLUSION

This paper focuses on *Responsive Sleeping*, where data from multiple sensors attached to the phone are continuously sampled and processed, even when the phone appears to be in the sleep mode. RS is an advanced feature that enables novel user interface, participatory sensing, and health and fitness applications.

Through detailed measurements we show that the current phone architecture, where all sensors are directly controlled by the phone processor, cannot meet the battery lifetime requirements for RS.

Based on these results, we propose a new phone sensing architecture for energy efficient responsive sleeping (EERS) on phones, where the sampling and processing of sensor data is offloaded to a low-power sensor processor. Using a step counting application running on the LittleRock prototype, we show that the proposed sensing architecture enables energy efficient *Responsive Sleeping* on phones.

8. REFERENCES

[1] B.Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J.-C. Herrera, A. Bayen, M. Annavaram, and Q. Jacobson. Virtual trip lines for distributed privacy preserving traffic monitoring. In *Mobisys'08*, June 2008.

[2] M.-C. Chiu, S.-P. Chang, Y.-C. Chang, H.-H. Chu, C. C.-H. Chen, F.-H. Hsiao, and J.-C. Ko. Playful bottle: a mobile social persuasion system to motivate healthy water intake. In *Ubicomp '09*, 2009.

[3] S. Consolvo, P. Klasnja, D. W. McDonald, D. Avrahami, J. Froehlich, L. LeGrand, R. Libby, K. Mosher, and J. A. Landay. Flowers or a robot army?: encouraging awareness & activity with personal, mobile displays. In *UbiComp '08*, 2008.

[4] R. C. Foster, L. M. Lanningham-Foster, C. Manohar, S. K. McCrady, L. J. Nysse, K. R. Kaufman, D. J. Padgett, and J. A. Levine. Precision and accuracy of an ankle-worn accelerometer-based pedometer in step counting and energy expenditure. *Preventive Medicine*, 41(3-4):778 – 783, 2005.

[5] J. Lester, T. Choudhury, and G. Borriello. A practical approach to recognizing physical activities. In *In Proc. of Pervasive*, pages 1–16, 2006.

[6] J. Lester, C. Hartung, L. Pina, R. Libby, G. Borriello, and G. Duncan. Validated caloric expenditure estimation using a single body-worn sensor. In *Ubicomp '09*, 2009.

[7] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao. Energy-accuracy trade-off for continuous mobile device location. In *MobiSys'10*, June 2010.

[8] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell. Soundsense: Scalable sound sensing for people-centric sensing applications on mobile phones. In *Mobisys'09*, June 2009.

[9] J. Paek, J. Kim, and R. Govindan. Energy-efficient rate-adaptive gps-based positioning for smartphones. In *MobiSys'10*, June 2010.

[10] Y. Wang, M. A. Jialiu Li and, Q. Jacobson, J. I. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *MobiSys'09*, June 2009.

[11] O. Woodman and R. Harle. Pedestrian localisation for indoor environments. In *UbiComp '08*, 2008.

[12] Z. Zhuang, K.-H. Kim, and J. P. Singh. Improving energy efficiency of location sensing on smartphones. In *MobiSys'10*, June 2010.