Little Rock: Enabling Energy Efficient Continuous Sensing on Mobile Phones

Bodhi Priyantha, Dimitrios Lymberopoulos, and Jie Liu
Networked Embedded Computing Group
Microsoft Research
Redmond, WA
{bodhip, dlymper, liuj}@microsoft.com

ABSTRACT

Although mobile phones are ideal platforms for continuous human centric sensing, the state of the art phone architectures today have not been designed to support continuous sensing applications. Currently, sampling and processing sensor data on the phone requires the main processor and associated components to be continuously on, creating a large energy overhead that can severely impact the battery lifetime of the phone. In this paper, we present the design and implementation of Little Rock, a novel sensing architecture for mobile phones, where sampling and, when possible, processing of sensor data is offloaded to a dedicated low-power processor. This approach enables the phone to perform continuous sensing at a low power overhead. We highlight and discuss in detail various design choices, tradeoffs and lessons learned. Using a pedometer application as an example, and by integrating Little Rock into an actual phone, we show that the proposed sensing architecture can be three orders of magnitude more energy efficient compared to the normal approaches.

1. INTRODUCTION

The ubiquity, mobility, and connectivity of cell phones make them an ideal platform for human-centered sensing applications. Phones today follow their users to almost every single activity they engage during the course of the day. In addition, current high-end smartphones have already been transformed to complex sensing devices since they have a rich set of built-in sensors (e.g. accelerometer, light, compass, pressure sensors and more) and a powerful processor for information processing. This allows phones to *continuously sense* their users and the environment they interact with, *understand* this environment and use this understanding to *provide meaningful services*. Several recent research projects, such as participatory sensing [6], sound sensing [12], and traffic sensing [5], have already used cell phones as the key sensing components in their systems.

However, continuous sensing, required by many phone-based applications, is challenging under the current phone architecture. Take an elderly assistance scenario as an example. A phone carried by an elderly person can be used as a pedometer to monitor her exercise and send daily reports to her health database. In addition, it can also detect when the person has fallen on the floor, and call emergency assistance immediately. A smart phone equipped with an accelerometer has the necessary capabilities to serve the purpose. But the challenge is on battery life. Designed mainly for *bursty* user interaction, current smart phones use the main processors to control the sensors directly. Continuous sensing implies that the main processor has to stay on all the time. These processors typically consume hundreds of mW when they are active (c.f. section 2) even when the screen and radios are not on. As a result, continuous sensing applications drastically reduce battery lifetime into a few hours, jeopardizing the usability of the phone.

One may think that dynamic voltage and frequency scaling techniques (DVFS) on phone processors have already solved processor energy management problems. However, due to the complexity of the processors used in today's mobile devices, static power consumption of the processor remains high (approximately 200mW whenever the processor is not in the sleep mode). In addition, for the correct operation of the phone several other components have to be operational, increasing phone's overall power consumption.

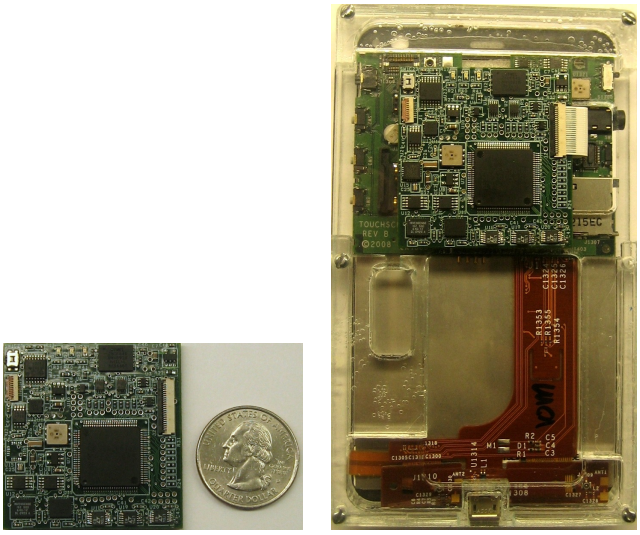
Another approach is to use an separate sensing module that employs its own sensors, processor and a wireless radio for interfacing to the phone. For example,

[10, 14] uses a bluetooth radio to interface a powerful sensor board to the phone. This approach is ideal when the sensor has to be placed on a specific location due to physical properties. For example, EKG sensors have to be attached to human body. On the other hand, continuous bluetooth communication can be energy consuming, and the main processor still needs to wake up often to exchange bluetooth packets. In addition, the user has to manage, carry and charge multiple devices which can be cumbersome.

In this paper, we explore the direction of building a small, energy efficient co-processor into the phone, and offloading continuous sensing tasks to the small processor. All the available sensors on the phone are connected to the small processor enabling the phone to transition to sleep mode while the co-processor is continuously acquiring and processing sensor data at a low power overhead. Since the two processors are tightly integrated, data between them can be exchanged fast and on demand because the small processor can wake up the main processor at any time and the main processor can request access to sensor data acquired by the small processor whenever it needs to.

Designing such an architecture is challenging in several fronts. First, the *low power operation* of the architecture is critical. The additional hardware components introduced to the phone (co-processor, sensors and supporting circuitry) must have the minimum possible impact on the power signature of the phone to ensure long battery lifetime. Second, the design should have *minimum impact* on other aspects of the phone design. Modern phones are built from OEM modules to take advantage of mass production efficiency. It is desirable not to break this design model. Third, the ability to *configure and reprogram* the sensing architecture is vital. Its role might change over time and therefore the phone should be able to easily reconfigure or even reprogram the core functionality of the sensing architecture.

Working towards addressing these challenges, we have designed and implemented *Little Rock*, shown in Figure 1, a sensing platform that can be seamlessly integrated into a smartphone. Little rock is built around an MSP430F5438 processor and a number of sensors including accelerometer, gyroscope, compass, pressure and temperature sensors. Our sensing platform consumes 12.9 mW when active, approximately 60 times less energy compared to the main processor on a typical smartphone. When the phone is in the sleep mode the combined power consumption of the phone and the *Little Rock* is only 7.87mW. By integrating *Little Rock* into an actual smartphone (Figure 1(b)) and implementing a sample continuous sensing application, we show that the proposed architecture can be three orders of magnitude more energy efficient compared to the current



(a) (b)

Figure 1: (a) The *Little Rock* sensing platform (b) The *Little Rock* board attached to one of our prototyping smartphones.

approaches while not significantly shortening phone's battery lifetime.

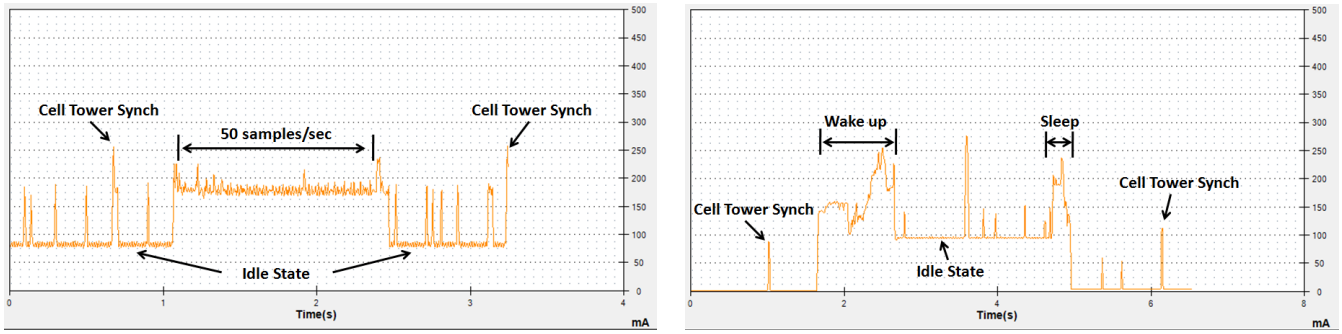
The rest of the paper is organized as follows. Section 2 highlights the major bottlenecks of the current phone sensing architectures and motivates the need for offloading sensor sampling and processing from the main processor of the phone. In Section 3 we introduce the Little Rock architecture and describe in detail its design, implementation and ability to be seamlessly integrated into and configured by a typical phone. In Section 4 we evaluate the proposed architecture using a simple continuous sensing application and compare the energy efficiency of Little Rock to that of a high end smart phone. Section 5 presents the related work and Section 6 concludes the paper.

2. MOTIVATION

Battery lifetime 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 lifetime of the phone. Consequently, enabling continuous sensing on mobile phones requires that both, the hardware sensors required and the necessary software for collecting and processing sensor data, have the minimum possible impact on the power signature of the phone. Table 1 shows the overhead introduced by popular types of sensors in the power consumption of a typical smartphone, the HTC Touch Pro running Windows Mobile 6.1. The power overhead for every sensor is expressed as a percentage of the power consumed by the HTC phone in 3 representative power

Sensor	HTC Touch Pro State		
	Active (1680mW)	Idle (399mW)	Sleep (7.56mW)
Accelerometer (0.56mW)	0.03%	0.14%	7.4%
Temperature (0.21mW)	0.0125%	0.053%	2.78%
Barometer (1.68mW)	0.1%	0.42%	22.2%
Compass (2.24mW)	0.13%	0.56%	29.63%
Total	0.2725%	1.173%	62.01%

Table 1: Overhead of different types of popular sensors on the overall power consumption of an HTC Touch Pro phone in 3 representative power states.



(a) (b)

Figure 2: 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.

states: *Active*, *Idle*, and *Sleep*. 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 phone 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* and *Idle* states increases by approximately 0.3% and 1% respectively. In the case of the *Sleep* state the combined power consumption of the phone and the sensors is approximately 12mW, which is comparable to the active power consumption of the processor in the MicaZ low power sensor node architecture.

Even though the continuous operation of the hardware sensors comes at a very small power overhead, the process of accessing and processing sensor data on current state-of-the-art phones might be 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 demonstrate this, consider the power consumption traces acquired using an HTC Touch Pro phone when the phone is in *Sleep*, and *Idle* states as well as when the phone is continuously sampling the built-in accelerometer at a rate of 50 samples per second (Figure 2). Note that 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 wake up in order to acquire and store every single accelerometer sample. In practice, this means that for every sensor sample 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 2. 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 full second. However, when continuous sampling is required even at very low sampling rates, 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 continuous sensing the phone must be

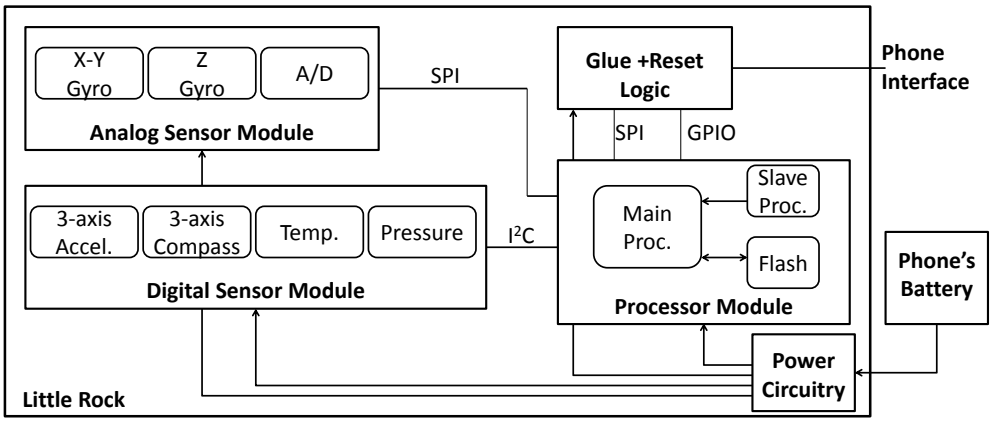


Figure 3: Overview of the *Little Rock* architecture

constantly on consuming approximately 756mW. Given a typical smartphone battery capacity of 1340mAh, an HTC Touch Pro that continuously samples its accelerometer would last for approximately 6.7 hours, without taking into account any real usage of the phone (phone calls, sms send/receive, 3G data traffic etc.). When considering the actual usage of the phone, the battery lifetime could be easily reduced to 3 hours or less depending on the particular type of user.

To address these challenges, we introduce the Little Rock architecture. Little Rock enables energy efficient continuous sampling on phones by offloading sensor sampling and processing on a low power processor. By decoupling the main processor of the phone from the sensors, we enable most parts of the phone to move to a low power *Sleep* state, while the low power processor is continuously sampling and processing sensing data at a low power overhead. The next section describes in detail the Little Rock architecture.

3. LITTLE ROCK

Figure 3 provides an overview of the *Little Rock* architecture. A low power processor is directly interfaced to a set of digital and analog sensors over the I2C and SPI serial interfaces. The sensor data acquired by the processor can be either stored immediately on the onboard flash memory or first filtered and processed by the main processor.

We highlight four features in this design:

Transparency: In the case where the phone needs direct access to one of the sensors on *Little Rock*, the main processor will act as a bridge between phone's processor and the actual sensor, enabling phone's processor to directly access any sensor through the SPI bus.

Power independence: *Little Rock* is powered directly from the battery and not from the internal power electronics of the phone. Thus, the majority of the main power circuitry can be turned off when the phone is in the sleep mode, and *Little Rock* can continue to be functioning, a key requirement for continuous sensing.

Interrupt: There might be cases where the sensing data collected by *Little Rock* requires that a specific service or action on the phone be triggered. For instance, in the case of the elderly assistance application mentioned in Section 1, when the system detects that the elder person has fallen, the phone might have to make an emergency call or send an e-mail. In order to achieve this functionality, *Little Rock's* main processor is able to interrupt and wake up the phone using a GPIO pin. The phone is then able to recognize the source of the interrupt and query *Little Rock* to identify the exact reason of the wake up event.

Re-purposing: *Little Rock* has two processor, a main and a slave.The secondary slave processor can be used by the phone to re-program the *Little Rock*'s main processor. This functionality can be particularly useful when the user installs a new sensing application on her phone, that requires a very specific "device driver" or types of processing on the sensor data. In this scenario, the phone can leverage the secondary processor to reprogram the main processor on the *Little Rock* board and enforce the new application's processing requirements.

In the rest of this section, we describe the *Little Rock* architecture and discuss the major design tradeoffs in detail.

3.1 Building Components

The Little Rock platform consists of four functional modules: the processor, digital sensor, analog sensor and phone interface components.

3.1.1 Processor Module

The processor module consists of an MSP430F5438 processor with 16kB RAM and 256kB flash memory. The current version of the processor can be clocked up to 18MHz. This processor also supports larger number

Function	Component number	Operating current μ A	Sleep current μ A	Manufacturer
3-axis accelerometer	BMA150	200	1	Bosch
Pressure sensor	BMP085	600	0.1	Bosch
3-axis compass	HMC5843	800	2.5 (dual supply) 110 (single supply)	Honeywell
Temperature Sensor	STTS75	75	1	ST Micro
X-Y axis Gyroscope	IDG-500	7000	7000	Invensense
Z axis Gyroscope	ISZ-500	4500	4500	Invensense

Table 3: Properties of various sensors on Little Rock

Name	Desciption
SPI	4-wire SPI bus with the phone as the bus master
GPIO	A phone processor GPIO pin. Can be configured as input or output. Can be used to interrupt the phone
VIO	1.8V supply for the internal phone IO.
PowerSW	A pin similar to phone's power switch. Conecting this pin to ground simulates pressing the power button.
VBATT	Phone's battery voltage
Ground	Ground connection

Table 2: Details of the phone expansion connector

of parallel and serial IO, enabling us to attach additional sensors, other than those that are already built in to Little Rock. This module also has a smaller MSP430 processor (MSP430F2013) for programming the main processor through the attached mobile phone. The processor module also contains an 8MB flash storage, and a potential divider for measuring phone battery voltage.

3.1.2 Digital Sensor Module

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 by an I2C bus. These sensors are powered from a separate voltage regulator to reduce the impact due to digital switching noise from the processor. Table 3 lists the main characteristics of the sensors on Little Rock.

3.1.3 Analog Sensor Module

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 converter to digitize the gyroscope output. To enable simultaneous sampiing of all 3 Gyroscope channels and minimize the data acquisition delay we used 3 separate A/D converts.

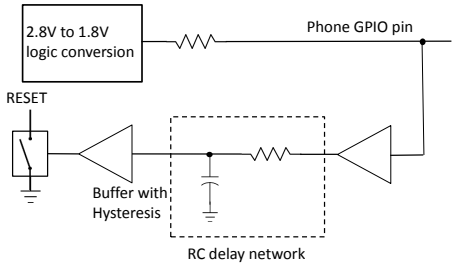


Figure 4: A single phone GPIO is used for interrupting the phone and to reset the main MSP processor

3.1.4 Reset and Wake Up Logic

Little Rock is able to interface to a phone through an expansion connector that includes the necessary signals for interfacing external circuitry to the phone. Table 2 shows the wired connections available on the expansion connector of the prototyping phone we used to interface to *Little Rock*. Note that besides the SPI bus and the power and ground lines, there is only 1 GPIO pin available in the interface between Little Rock and the phone. However, we need at least two control mechanisms between the two units. First, we need be able to reset the MSP430 from the phone, and second, the MSP430 should be able to interrupt the phone. We used the circuit shown in Figure 4 to achieve both of these functions with a single GPIO signal. Under normal operation, the phone configures the GPIO pin as an interrupt input signal. When the processor wants to interrupt the phone, it drives the GPIO pin signal low for $5\mu s$. The RC delay network prevents this short pulse from driving the reset pin of the MSP430 low. When the phone wants to reset the MSP430, it configures the GPIO pin as an output and generates a 500ms logic low signal. This duration is enough to create a logic 0 at the output of the RC network, driving the RESET signal of the MSP430 processor low using an open collector driver.

3.2 Phone Interface

Even though *Little Rock* can acquire, process and store sensor data independently of the phone's state, in many cases the phone might require direct access to the sensors or the MSP430 processor itself. *Little Rock* has been designed so that the MSP430 processor can be: (i) used as a bridge between the phone's processor and the sensors and (ii) reprogrammed by the phone;s processor.

3.2.1 Directly Accessing Sensors

The MSP430 processor on *Little Rock* is interfaced to the phone through the SPI bus, while the MSP430 connects to digital sensors through the I2C bus. When the phone needs to access a sensor directly, reading the accelerometer for example, the MSP430 acts as a bridge between the phone and the sensor, converting messages on the SPI bus to I2C messages and vise versa. However, the necessary bridging of the SPI and I2C buses introduces delays that could lead to erroneous data transfers between the phone and the sensors. These delays are introduced due to 2 main reasons. First, the I2C bus operates at a maximum clock speed of 400kb/s, while the SPI can operate up to 18Mhz (maximum processor clock). Due to this speed mismatch, the MSP430 has to buffer the messages received over the SPI as they are being sent over the I2C; similarly the results received over the I2C have to be buffered so that the received bytes can be sent as a continuous byte stream message on the SPI bus. This buffering results into delays inthe communication between the phone's processor and the sensors. Second, unlike the SPI bus, devices with an I2C interface have the option to delay the bus activity by holding the clock like low (clock stretching), if they cannot respond immediately to a message. This can result in a variable response time when accessing a device over I2C which of course could affect the timing of the communication between the MSP430 and the phone's processor over the SPI bus. Since the SPI slave does not have control over SPI bus transmissions, the only way for MSP430 to address these delays is to insert dummy bytes of value 0xFF while these delays are taking place. When the response is ready, MSP430 sends an ACK byte (value 0x55) followed by the response. After receiving the ACK, the master, in this case the phone's processor, reads the actual data byte or bytes returned by the sensor.

3.2.2 Phone Software API

We developed a kernel driver with an *IO control* (ioctl) interface for accessing *Little Rock* from user applications on the phone. This interface provides methods to transfer a byte array over the SPI bus, to set the GPIO pin direction, to set and read the GPIO pin value, to control interrupts on the GPIO pin, and to register GPIO interrupt event handlers.

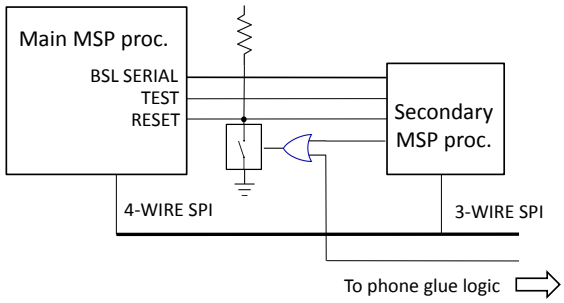


Figure 5: Secondary and main MSP430 processor connections for reprogramming the main processor

3.2.3 Reprogramming Support

To reprogram *Little Rock's* main processor, we use a secondary MSP430 processor and the built in BSL routines of the main processor. Although we could have developed our own reprogramming routine, which kicks in each time the MSP430 processor is reset to receive reprogramming commands and data over the SPI bus, we decided against this for two reasons. First, we decided to use the well debugged built in routines rather than our own routine because we want users with limited MSP430 experience to start using our platform. Requiring the users to return the phones to us or expect them to reprogram the MSP430 using the JTAG interface, due to any bugs that may be present in our debug routine, would create a negative user experience. Second, we anticipate that the secondary processor may be useful in the future for housekeeping operations, such as acting as an intelligent watch dog. Figure 5 shows how the main and secondary processors are connected. The secondary processor shares the SPI bus that connects the main MSP processor and the phone. However, since SPI is a point-to-point bus, the secondary processor makes sure that its SPI interface is disabled whenever the main processor's SPI interface is active. The phone initiates the reprogramming of the main processor by asserting its RESET pin and transmitting a special command message over the SPI bus. This message is not received by the main MSP430 SPI interface since the processor RESET pin is asserted. The secondary processor detects the assertion of the REST pin through a GPIO interrupt, and starts decoding SPI data sent by the phone by emulating the SPI protocol in software. The secondary processor keeps its SPI output pin in high impendence mode. If the RESET pin is released without receiving a reprogramming message from the phone, the secondary

processor abandons the reprogramming and returns to sleep.If the secondary processor receives a reprogramming command on SPI bus while the RESET is asserted, it waits until the RESET pin is de-asserted, activates its SPI interface and invokes the main MSP's BSL routine by manipulating RESET and TEST pins. During reprogramming, the secondary processor acts as a bridge between the phone and the main MSP430 processor. The secondary processor emulates RS232 protocol in software to communicate with the main processor's serial BSL interface.

3.3 Enabling Low Power Operation

Since the *Little Rock* board is powered directly from the phone battery , it is essential to make sure that is impact on the power signature of the phone and therefore its impact on the phone's lifetime is minimal.

3.3.1 Isolation at the Glue Logic

Providing proper isolation is critical for both eliminating excessive current as well as preventing hardware failures. Similar to using correct logic values at processor GPIOs, we have to provide proper isolation between the phone and the MSP430 when the phone is turned off. As an example, any logic input to the phone should be disabled as soon as the phone GPIOs interfaces are turned off; otherwise this can result in excessive current going in to the turned off circuits on the phone, that besides increasing power consumption, it could cause hardware damage on the phone. Instead of using separate isolation logic, we achieved this isolation within logic gates used for converting the 1.8V and 2.8V logic levels between the phone and the MSP430. We used logic gates with *power off* capability as logic level converters. The *power off* capability states that when the power to a logic gate is turned off, both the inputs and the outputs of that logic gate act as open (tri-stated) connection with very small leakage current, irrespective of the logic level applied to that pin. Some of the level converters we used were completely or partially powered from the 1.8V GPIO supply of the phone. When this supply is deactivated, due to phone going to sleep or the phone being turned off, the appropriate logic signals on the interface are tri-stated to prevent excessive current or hardware damage.

3.3.2 Reducing Gyroscope sensor sleep currents

Most of the sensors used in *Little Rock* support low power sleep modes (Table 3). Gyroscopes, however, do not have a low power mode; together they consume $\simeq$11.5mA when idle. In addition, a reference voltage buffer used with the Gyros consumes $\simeq$2mA when idle. To reduce this idle power we turn off the power supply to the Gyros and the A/D converters at the gyro outputs. Although the A/D converters have a low power mode that we could leverage, we connect them to the same power supply as the Gyros to reduce noise. To prevent excessive currents flowing in to powered off Gyroscopes subsystem, we configured the GPIO pins of the MSP430 processor as inputs.

Even when the Gyro power is switched off, and the GPIOs configured as inputs, we continued to observe several hundreds of mV at multiple GPIO pins connecting to the Gyro module. When the GPIO pins were physically disconnected from the Gyros, we observed that the sleep current of the processor module dropped by a factor of 2. When the GPIO pins were connected, we could not achieve the same low sleep current irrespective of how we configured the GPIO pins (activating pull down resistors, for example). Finally, we decided to add analog switches to isolate the GPIO pins. Since there are low power analog switches that only consume $\simeq 1\mu$A we managed to reduce the overall sleep current.

3.3.3 Reducing compass sensor sleep currents

The 3-axis compass IC we selected supports two different power supply options. In one option, the analog section is supplied with 2.8V while the digital section of the IC is supplied with 1.8V. In the other option, an external 2.8V is applied and an internal regulator supplies the 1.8V digital power. Since *Little Rock* only has a 2.8V internal supply, we were tempted to use the single supply option.

However, according to the datasheet, as well as our own measurements, the single signal supply option has much higher sleep current of $\simeq 100\mu$A (even higher than the rest of the *Little Rock* board) compared to the dual supply option, which consumes $\simeq 2.5\mu$A. Instead of using the inefficient internal regulator of the compass sensor, we used a separate low power regulator to supply the required 1.8V supply at a much lower overhead than with the built-in compass regulator. In this case also, adding more components resulted into reducing the overall system sleep current.

4. EVALUATION

In this section we evaluate the *Little Rock* architecture in detail. First, we provide a detailed breakdown of *Little Rock's* power consumption at different operating modes. Next, we integrate *Little Rock* into an actual prototyping phone and use a pedometer application that is continuously sampling the accelerometer sensor to compare the performance and energy efficiency of the proposed architecture to the current sensing approaches on mobile phones.

4.1 Little Rock's Sleep Power Profile

Since typical sensing applications involve heavy duty

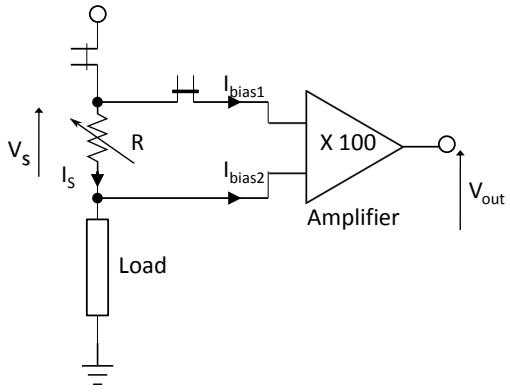


Figure 6: Setup for measuring sleep currents

cycling of the processing and sensing modules, the sleep currents have a big impact on the average power consumption. For instance, with a sampling rate of 10 samples per second, *Little Rock* will be in sleep mode most of the time. Therefore, minimizing *Little Rock's* sleep power consumption is critical.

Since the sleep currents of the different components on the *Little Rock* board are typically in the order of μA, accurately measuring these currents immediately became a challenge. As a result, we first describe our power measurement setup that enabled us to accurately measure small currents and then show how we used our setup to power profile *Little Rock*.

4.1.1 Measuring small currents

Although Prabal et al. presents a solution for measuring small static and dynamic currents, their solution depends on the use of switching regulators [8]. However, this approach cannot be used in the case of *Little Rock* because it does not use switching regulators due to the noise concerns.

The setup we used to measure small currents can be seen in Figure 6. The variable resistor, R acts as the current sensing resistor generating a voltage V_s proportional to the current I_s. Since R is interposed between the voltage supply and the load, V_s should not be larger than several mV (we limit V_s to $10mV$). Because of the small value of V_s and the various electromagnetic interference, accurately measuring V_s is a difficult task. To address this challenge, we used a common setup that is used for current sensing [3] and employs an amplifier to amplify V_s.

However, we still face two challenges when measuring small sleep currents. First, to generate several mV from several mA, we need a large resistor R (in order of $k\Omega$s). However, processors and sensors consume a much larger current (several mA) at the startup, before they go into sleep mode. So, a large R could cause a voltage drop of several volts at the startup, preventing these components from starting due to low supply voltage.

Module	Sleep current (μA)
Processor + glue logic	36
Digital sensors	29
Analog sensors	2
Total	67

Table 4: Current dissipation for the different functional components of *Little Rock* in sleep mode.

We solve this by using a variable resistor. At the start we set R to the lowest value (0 ohms). After the load enters the sleeping state, we increase R until V_{out} reaches $\simeq 1V$, corresponding to a 10mV drop across R. Then we disconnect the jumpers and measure R using a multi-meter. We obtain the I_s by $I_s = \frac{V_{out}}{100 \times R}$.

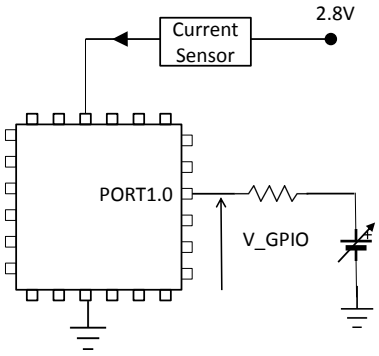
The second challenge is that the input current flowing in to the amplifier input itself (I_{bias1} and I_{bias2}) has to be much smaller compared to I_s to measure I_s accurately. We use an instrumentation amplifier with switched capacitor inputs for the amplifier [11]. The input currents of this amplifier are in the order of several nA, making it suitable for accurately measuring currents of several μA.

4.1.2 Little Rock Sleep Currents

Having an accurate measurement setup allowed us to profile in detail the sleep currents of various submodules in Little Rock. We use three 2.8V linear regulators to supply power to different Little Rock modules. One regulator supplies power to the processor and glue logic modules. To reduce power supply noise, we use another regulator to power the sensors with digital interfaces, a third regulator, which can be turned off by the processor, supplies power to the Gyroscope module. Table 4 shows the sleep mode currents at the input of these three regulators. With a 1340mAH battery capacity, the Little Rock can operate up to 830 days while in sleep mode. Given the typical phone battery life is a couple of days and the fact that a typical phone's sleep current is in the order of $2mA$, the Little Rock, when in sleep mode, has little impact on the overall battery lifetime.

Little Rock's low sleep current dissipation was mainly achieved by the design decisions described in Section 3.3. To demonstrate this, we show how the sleep current of the processor is affected by external voltages applied to its input GPIO.

We use the setup in Figure 7 to evaluate the impact of external voltages to input GPIO pins on the sleep current of the processor. We use a MSP430F5438 evaluation board. We power the processor using a 2.8V supply and attach the current sensor described in the previous



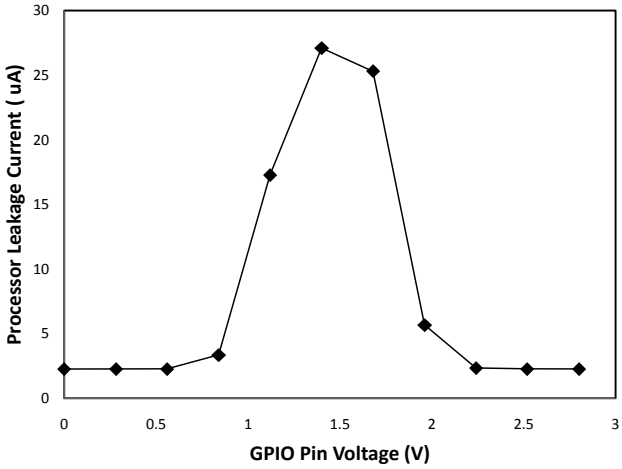


Figure 7: Setup to measure the impact of GPIO voltage on the processor sleep current. The MSP430F5438 GPIO pin PORT1.0 was configured as an input, and a variable voltage was applied to it. All the other GPIUO pins were configured as outputs.

Figure 8: Processor Sleep current vs. the GPIO voltage on Port1.0 GPIO pin. Other GPIO pins are configured as output and set to logic 0.

section to measure processor current. We configure all the pins, except pin PORT1.0, of the MSP processor as logic 0 outputs, and we put the MSP430 in to sleep (low power mode 4). Next we apply a variable voltage source, connected through a 10kΩ to the pin PORT1.0.

Figure 8 shows the processor's sleep current as a function of the voltage at the PORT1.0 processor pin. When the GPIO pin is at 0V and 2.8V, we observed the minimum sleep current of 2.26μA. When the GPIO voltage reaches 1.4V, the sleep current increases by more than factor of 10 to 27.1μA. It is clear that extreme caution should be taken when voltages levels, other than those close logic 1 or 0 are applied to processor pins that are configured as inputs. There are multiple causes for such voltages, such as, a processor D/A converter input with an analog voltage is configured as a digital input, electromagnetic noise-induced stray voltages on unconnected processor pins, or , as in our case, voltages induced on pins attached to a powered down submodule.

4.2 Pedometer Application: A Case Study

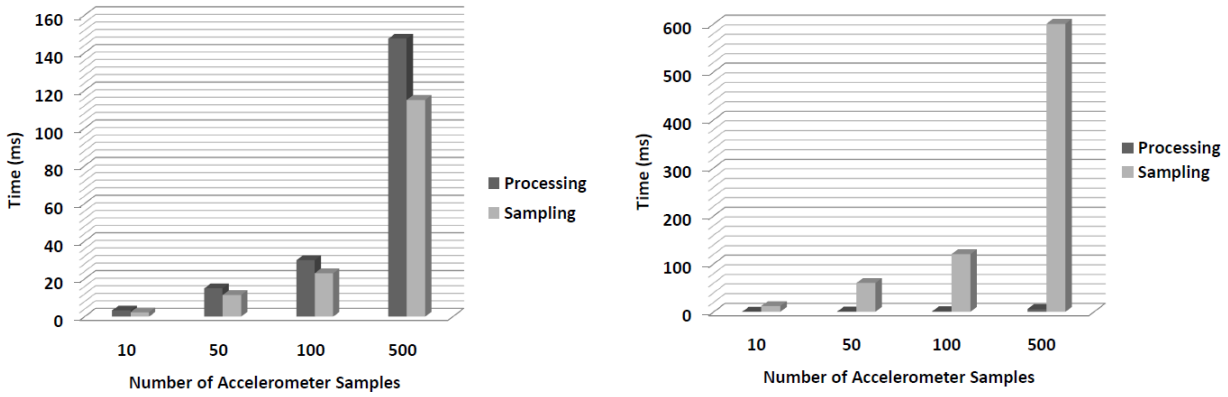
In this section, we evaluate the performance of running a continuous sensing-based application on *Little Rock*, on a phone, and on a phone equipped with the *Little Rock* board (shown in Figure 1(b)). We use the step counting application shown in Listing 1 as our example application. This application samples the 3-axis accelerometer at a frequency of f samples/sec, and after every n samples, it executes the routine in Listing 1. This routine will analyze the latest n samples to identify how many steps were performed by the user carrying the phone.

Platform	Little Rock	Phone
Latency per accelerometer sample (ms)	0.23	1.2
Energy per accelerometer sample (mJ)	0.008	0.91
Active Power (mW)	12.9	756
Sleep Power (mW)	0.27	7.6
Wakeup transition time (ms)	2.5μA	900
Wakeup transition energy (mJ)	$\simeq$0	624
Sleep transition time (ms)	$\simeq$0	270
Sleep transition energy (mJ)	$\simeq$0	187

Table 5: Latency and energy cost of Little Rock and the phone for several primitive operations

4.2.1 Basic Platform Profiling

Table 5 shows the cost of basic operations such as the cost of collecting individual accelerometer samples and the cost of transitioning between the active and sleep modes for both the *Little Rock* board and the phone. Note that the energy required to acquire a single accelerometer is approximately 2 orders of magnitude lower than the energy required by the phone. This is due to two main reasons. First, *Little Rock*'s power consumption when sampling the accelerometer is much lower than the power consumption of the phone (12.9mW to 756mW). Second, even though the phone's processor is much faster than the processor on *Little Rock*, the phone is 5 times slower than *Little Rock* in terms of acquiring a single accelerometer sample. This is due to the fact that the processor on the phone has to continuously support a very complex software stack. At any time, the operating system might be executing several background tasks or services that might slow down the



(a) Little Rock (b) Phone

Figure 9: The overhead for acquiring and processing accelerometer data as a function of the number of samples.

performance of the processor. In addition, in order for an application to access the accelerometer, a number of function calls have to be made throughout the whole operating system software stack that might increase the overall delay.

Table 5 also shows that the sleep mode and transition energy overheads of *Little Rock* are much lower compared to the phone. The sleep power of the phone is approximately 28 times higher than that of *Little Rock*, and *Little Rock* can almost instantly switch between active and sleep modes.

```
1  void ProcessAccelerometerData() {
       double lastMag = magLast;
3      double maxMag = magMax;
       double currMag = 0;
5    for (int i = 0; i < n; i=i+3){
       currMag = Math.Sqrt(
7        (acceldata[3*i]*acceldata[3*i]) +
         (acceldata[(3*i)+1]*acceldata[(3*i)+1]) +
9        (acceldata[(3*i)+2]*acceldata[(3*i)+2]));
       if (!seenBottomInflection) {
11       seenBottomInflection = currMag > lastMag;
       }
13     lastMag = currMag;
       if ((maxMag - currMag > accel_threshold) &&
15           seenBottomInflection){
         numSteps += 1;
17       maxMag = currMag;
         seenBottomInflection = false;
19     } else if (currMag > maxMag) {
         maxMag = currMag;
21     }
     }
23   magLast = currMag;
     magMax = maxMag;
25 }
```

Listing 1: An accelerometer-based step counting application.

Even though *Little Rock* consumes significantly lower power compared to the phone, the phone has significantly more processing power. Figure 9 shows the time it takes for both *Little Rock* and the phone to continuously acquire a number of accelerometer samples and process them using the code shown in Listing 1 for different number of samples. It is apparent that the bottleneck in the case of *Little Rock* is processing. The time it takes to process the data is always higher than the time to acquire the data and as the number of samples increases the difference also increases. On the other hand, phone's major bottleneck is the time to acquire accelerometer samples, which can be orders of magnitude higher than the actual processing time.

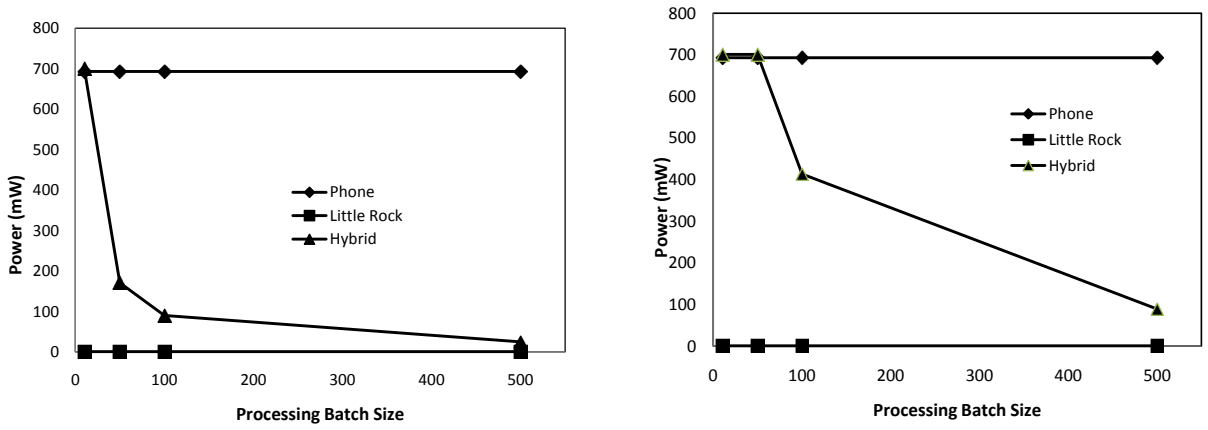
4.2.2 *Energy Efficient Pedometer*

To provide a more realistic evaluation of the different platform configurations we consider in this paper, we implemented and profiled the pedometer application (Listing 1) while running on the phone, on *Little Rock* as well as on a hybrid architecture that includes the phone with an embedded *Little Rock* board. We profiled the pedometer application for all combinations of 2 different sampling rates (10 and 50 samples per second) and 4 processing batch sizes[1] (10, 50, 100, 500). At every run, the pedometer application will execute until 500 samples have been collected and successfully processed. In the case of the hybrid architecture, *Little Rock* is responsible for sampling the accelerometer but whenever processing of the data is required, it will wake up the processor, transfer the data and let the phone do the processing.

Figure 10 shows the average power consumption for all three different hardware configurations. Phone seems to be the most energy inefficient configuration since it consumes on average approximately 700mW. Note that this number does not change for different sampling data rates because in all cases the processor on the phone has to be constantly on to timely acquire accelerometer samples.

On the other hand, *Little Rock* consumes on average 0.7mW (50 samples per second) or 0.2mW (10samples per second) depending on the sampling data rate used.

[1]A processing batch size is the number of sensor samples we have to collect before we start processing the data.



(a) 10 samples per second

(b) 50 samples per second

Figure 10: Average power required to acquire and process 500 samples for different sampling rates and processing batch sizes.

This corresponds to more than 3 orders of magnitude improvement in the power consumption of the phone. Note that the power consumption in the case of *Little Rock* changes with the sampling data rates used. Conversely to the phone, the MSP430 processor on *Little Rock* can almost instantly move to sleep mode when not sampling or processing data. Therefore, the lower the sampling data rate used, the more time it spends in sleep mode and therefore its power consumption is lower.

Figure 10 also shows that the hybrid approach can be very efficient if it is appropriately configured. For small processing batch sizes the power consumption is very similar to the one of the phone because the phone has to wake up very often to perform the processing of the sensor samples. However, when the processing batch size increases, the power consumption of the hybrid approach can be up to 70 times lower than the power consumption of the phone itself. The reason is that when higher processing batch sizes are used, the phone can spend more time in the sleep mode and thus, drastically reduce its power consumption.

5. RELATED WORK

Participatory sensing applications that use cell phones as a widely deployed sensors network have gained much popularity [6, 16, 1, 9, 17]. Some of these participatory sensing applications require continuous monitoring with cell phones. The Nericell [13] uses smart phones with built in sensors that continuously sense the environment to detect various road conditions. The authors use multi sensor-based triggers to reduce the energy consumption due to continuous sensing. Soundsense [12]) continuously monitors the audio signals using the phone's microphone to classify and identify various location and activity related events. We observe that these continuous monitoring applications can have a severe impact on the phone battery life, and we propose Little Rock as a general sensing architecture for mobile phones that enables energy efficient continuous sensing applications.

Another class of phone-centric sensing research uses a physically separate sensing platform, connected to a cell phone using a BlueTooth radio, for collecting human centric data. Lester et al. uses a body worn sensor platform with multiple sensors and a processor for activity recognition [10]. The sensor platform collects various sensory data and uploads the data to a cell phone using a BlueTooth radio; the activity recognition algorithms execute on the phone. The Health Gear project [14] uses a blood oximeter, which is attached to a sensing platform that communicates with a cell phone over BlueTooth for detecting sleep apnea. This work further highlights the need for continuous sensing for building rich, user centric, application on mobile phones. However, unlike these platforms which are point solutions for a particular application, we propose a novel sensing architecture for cell phones that enables building richer applications for cell phones.

Wireless sensor network platforms enable low power environmental sensing using custom built platforms and low power multi-hop wireless networks (refer the platforms). To overcome the limitations of such low power wireless networks, several research projects have used the cell phone network for data dissemination [4, 7]. While we acknowledge that cell phones may not replace low cost, low power wireless sensor nodes, our proposed architecture enables low power environmental sensing on cell phones.

Pering et al. developed the PSI board [15], a hardware platform with an accelerometer, a low-power MSP430

processor, and an 802.15.4 radio that can be plugged in to the SD card slot of a cell phone. Although the authors mention the energy savings due the MSP430 processor, this platform was meant as an expansion module that forms a bridge between the phone and a 80.15.4 wireless network. In contrast, we propose a generic sensing architecture for phones that off loads all background sensing tasks to a low power processor; we do a detailed comparison of energy consumptions when sensing is offloaded to an external processor vs. driving the sensors directly from the phone's application processor. Our work is similar to Somniloquy that off loads network packet processing to a low power processor [2].

6. CONCLUSION

Continuous sensing is a basic requirement for a class of mobile phone applications. Through detailed measurements we have shown that the current phone architecture, where all sensors are directly controlled by the phone processor, cannot meet the battery lifetime requirements of the phone. Through the design of *Little Rock*, we introduce a new module into the phone architecture, which can offload the interactions with sensors and give the phone's main processor and associated circuitry more time to go to the sleep mode. This can result in significant savings. For a pedometer application, the energy savings by running with *Little Rock* is three orders of magnitude compared to running on the current phone architecture.

The *Little Rock* architecture gives programmers more flexibility to choose where to allocate their applications, but it also brings challenges on application development. As future work, we will investigate how to provide tools and programming models to simplify software development.

7. REFERENCES

[1] T. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. Guibas, A. Kansal, S. Madden, and J. Reich. Mobiscopes for human spaces. *IEEE Pervasive COmputing*, 6(2):20–29, 2007.

[2] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta. Somniloquy: augmenting network interfaces to reduce pc energy usage. In *NSDI'09*, pages 365–380.

[3] Linear technology : Current sense circuit collection. http://cds.linear.com/docs/Application Note/an105.pdf.

[4] P. M. Aoki1, R. J. Honicky, A. Mainwaring, C. Myers, E. Paulos, S. Subramanian, , and A. Woodruff. Common sense: Mobile environmental sensing platforms to support community action and citizen science. In *UbiComp 2008*, Seoul, South Korea, September 2008.

[5] 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 *Proc. of 6th ACM Conference on Mobile Systems (Mobisys'08)*, June 2008.

[6] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava. Participatory sensingn. In *WSW '06*, Nov 2006.

[7] H. Dubois-Ferrière, R. Meier, L. Fabreand, and P. Metrailler. Tinynode: a comprehensive platform for wireless sensor network applications. In *IPSN '06*, pages 358–365, 2006.

[8] P. Dutta, M. Feldmeier, J. A. Paradiso, and D. E. Culler. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *IPSN '08*, pages 283–294.

[9] B. Hull, V. Bychkovsky, K. Chen, M. Goraczko, A. Miu, E. Shih, Y. Zhang, H. Balakrishnan, and S. Madden. CarTel: A distributed mobile sensor computing system. In *ACM SenSys*, 2006.

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

[11] Linear technology : Ltc6800 datasheet. http://cds.linear.com/docs/Datasheet/6800fas.pdf.

[12] 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 *Proc. of 7th ACM Conference on Mobile Systems (Mobisys'09)*, June 2009.

[13] P. Mohan, V. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *SenSys '08*, pages 323–336, 2008.

[14] N. Oliver and F. Mangas. HealthGear: A Real-Time Wearable System for Monitoring and Analyzing Physiological Signals. In *BSN '06*, April 2006.

[15] T. Pering, P. Zhang, R. Chaudhri, Y. Anokwa, and R. Want. The PSI Board: realizing a phone-centric body sensor network. In *BSN '07)*, March 2007.

[16] O. Riva and C. Borcea. The urbanet revolution: Sensor power to the people! *IEEE Pervasive Computing*, 6(2):41–49, Apr-Jun 2007.

[17] A. Yu, D. L. A. Bamis, T. Teixeira, and A. Savvides. Personalized awareness and safety with mobile phones as sources and sinks. In *UrbanSense08*, November 2008.