Wireless Body Area Networks: Where Does Energy Go?

Sangwon Lee[1], Murali Annavaram[2]
[1]Computer Science Department, [2]Ming Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089, USA
{sangwon.lee, annavara} @ usc.edu

ABSTRACT

Wireless Body Area Networks (WBANs) promise to revolutionize health care in the near future. By integrating biosensors with a mobile phone it is possible to monitor an individual's health and related behaviors. Monitoring is done by analyzing the sensor data either on a mobile phone or on a remote server by relaying this information over a wireless network continuously and in real time. However, the "wireless" aspect of WBAN is being limited by the battery life of the mobile phone. A WBAN designer has a range of options to trade-off limited battery with many important metrics. From the choice of programming languages to dynamically choosing between computation versus communication under varying signal strengths, there are several non-obvious choices that can have dramatic impact on battery life. In this research we use an in-field deployed WBAN called KNOWME to present a comprehensive quantification of a mobile phone's energy consumption. We quantify the energy impact of different programming paradigms, sensing modalities, data storage, and conflicting computation and communication demands. Based on the knowledge gained from the measurement studies, we propose an Active Energy Profiling strategy that uses short profiling periods to automatically determine the most energy efficient choices for running a WBAN.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design studies

Keywords

Wireless Body Area Networks, Energy efficiency, Mobile Sensing, Smartphone

1. INTRODUCTION

Wireless Body Area Network (WBAN) is an exciting technology that promises to bring health care to a new level of personalization. The size of a transistor continues to shrink following Moore's law, which in turn is allowing miniaturization of sensor nodes. Miniaturization is particularly attractive in the health domain as these sensors can be worn on the body and they can non-intrusively monitor a person's physiological state. Multiple sensors communicate with an external data collection node using wireless interfaces forming a WBAN. WBANs enable monitoring an individual's health continuously in free living conditions, where the individual is free to conduct his/her daily activity. Recently WBANs are employing mobile phones to collect health data from sensors, store and even partially process data locally, and transmit the health data over wireless links to a back-end processing server.

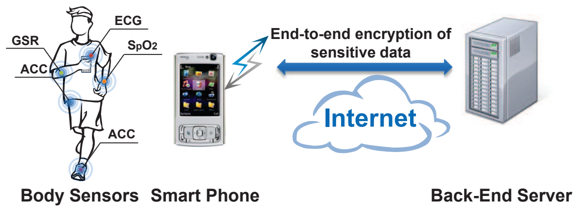


Figure 1: An Example 3-Tier WBAN System

A typical mobile phone based WBAN consists of three layers as shown in Figure 1. The first component is the sensor layer which measures physiological and even emotional signals and transmits this data wirelessly. The second layer is a mobile phone which acts as a data collection hub and receives the external sensor data. It may further enrich the sensor data with GPS , audio and video tags to get an accurate state of a person's health and environmental conditions. The mobile phone may also process data locally. The last layer is a back-end server that processes and stores the data. As WBANs operate continuously, the primary bottleneck to data collection and analysis is the limited battery life of mobile phones. For instance, in [34] the authors report that a Nokia N95 phone battery has nearly 200 hours of standby time, but when it is used in a WBAN setup, the battery drains in 4 hours using S60 Python. For WBANs to be used pervasively, it is necessary to first understand where the energy is spent in a WBAN and then use energy efficient strategies to operate a WBAN.

At each stage of a WBAN design and operation, the system designer has a wide range of options to trade-off battery life for other important metrics. During system software development phase the designer has a choice of programming platforms for improving software productivity. The designer may use Python, Java or a phone's native programming model such as Symbian C++ or iPhone SDK to develop the system software. The choice of language can tradeoff software productivity with potential runtime overheads of managed environments that lead to energy inefficiency.

The sensed data collected on the mobile phone can be buffered locally or transmitted to server immediately. Local buffering to flash is energy intensive. In fact, writing a small packet of data to flash storage may consume more energy per bit stored compared to transmitting over a 3G wireless link.

During the data transmission phase, the designer has to make energy tradeoff decisions on computation and communication costs. Storing data on flash may allow the mobile phone to compress large chunks of data and send compressed data to the remote server. Data compression places more compute demands. On the other hand, compression may reduce the transmission energy cost. Local storage of data also allows mobile phone to process the data locally and only send interesting events to the back-end. For instance, detecting an abnormal heart beat signal from ECG will reduce the need to continuously transmit normal operational data thereby reducing transmission energy.

As demonstrated with these ample examples, the system designer is faced with a daunting list of tradeoffs . While some previous studies have quantified the energy tradeoffs in specific domains, such as communication versus computation [1, 8], sampling rate versus accuracy [12, 13, 32], to the best of our knowledge, there is no published work that provides a comprehensive quantification of energy costs of the various choices a designer has to make. Thus the goal of this paper is two fold: first, we provide a comprehensive energy consumption evaluation of the various WBAN design choices using a Nokia N95 phone. We then compare the energy consumption of a few WBAN components across Nokia N95, E75 and Apple iPhone to demonstrate that the observations made on N95 are applicable to a broader set of mobile platforms. As a second goal, using the knowledge gained from the evaluations we propose Active Energy Profiling (AEP) strategy that uses short profiling intervals to automatically determine the best energy efficient operating point of a WBAN. We have gained access to KNOWME [3], a WBAN used for pediatric obesity monitoring, which we will use as a demonstration vehicle to highlight the energy consumption tradeoffs.

The rest of this paper is organized as follows: Section 2 describes KNOWME, the WBAN framework used in this study. Section 3 shows the experimental setup used to evaluate the effectiveness of our proposed architecture and presents results from our evaluations. We then make use of the observed energy consumption tradeoffs to describe and evaluate AEP in Section 4. Section 5 describes previous studies that are related to our work. Finally, we conclude in Section 6 and discuss future directions.

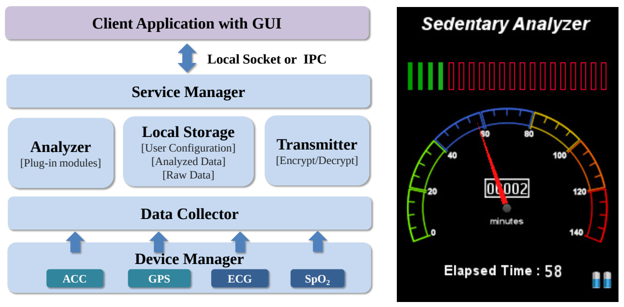
2. KNOWME PLATFORM

In this section, we provide details of the WBAN platform, called KNOWME, used in this study. KNOWME is a low cost mobile phone centric WBAN which is currently deployed in Los Angeles for pediatric obesity prevention and treatment. In the current implementation, each KNOWME node consists of a Nokia N95 mobile phone (N95) and three health sensors, namely tri-axial accelerometers (ACC), electrocardiograph sensor (ECG) , blood oxygen saturation sensor (OXI), and one location sensor, GPS. The ACC, ECG and OXI are Bluetooth enabled off-the-shelf sensors from Alive Technologies, while GPS is a built-in on N95. The ECG samples at 300Hz, OXI uses 100Hz sampling, ACC uses 30Hz sampling while GPS can be sampled at any user-specified rate.

2.1 KNOWME Software Component

KNOWME software runs as a mobile application on N95. The mobile application has two components corresponding: a background process (KMCore) for data collection in background and a client interface application (KMClient) for configuring the sensors and data visualization. The KMCore comprises of seven components arranged in a four-tier hierarchy: (1) device manager at the bottom-most tier (2) data collector at the second tier (3) at the third tier there are data analyzer, local storage manager, and data transmitter and (4) a service manager at the top tier. Figure 2(a) shows how various components in the KMCore interact each other. There is one thread per each sensor called device manager thread that receives data from its associated sensor. By creating a separate thread per each sensor KNOWME continues to get data from working sensors without being hindered by a single non-functional sensor. The data collector thread receives sensor data from each device manager and synchronizes all the sensor data with the same timestamp into a single health record. It combines several such records into a single write buffer and sends it to the local storage manager to write the data to the flash storage. When the flash storage runs out of space old data is replaced with new data in a FIFO manner.

The analyzer modules are designer defined modules that perform domain specific tasks. In pediatric obesity management domain these modules perform user state classification using multi-modal signal processing algorithms. In current implementation the analyzer classifies user state as either sedentary or non-sedentary using



(a) (b)

Figure 2: KNOWME Application: (a) The Components of KMCore (b) A Screenshot of KMClient Running on N95

just ACC data with Sedentary Analyzer (SA) based on Support Vector Machine classifier [14]. The transmitter module transfers data to back-end and handles data compression and encryption for privacy and energy saving.

The back-end server runs a more comprehensive suite of classification algorithms to detect a range of user states such as walking, running, fidgeting, standing, sitting using ECG and ACC sensor data with multi-modal signal processing [28]. Such finer classification is used by the physicians at the back-end to get a comprehensive understanding of user behavior and to precisely measure the calories burned. Back-end server stores sensor data indefinitely. In other words, the data stored on the back-end server is a complete record of a person's physical activity history, and the most recent time window of this history is stored on the mobile phone's flash storage.

The last component of the KMCore application is a service manager thread that uses sockets or inter-process communication (IPC) to provide the sensor data to other mobile applications running on the phone such as data visualization application, KMClient. Figure 2(b) shows a visualization screen of the KMClient that shows how long a user is sedentary from SA.

3. ENERGY IMPACT OF DESIGN CHOICES

Using the KNOWME framework described in the prior section, we will now provide a comprehensive evaluation of the energy consumption of the various components in KNOWME. While KNOWME is one particular implementation of a WBAN, we would like to note that most WBANs that we are aware of have very similar architecture [20, 23]. Furthermore, where possible we quantify the energy consumption of basic operations without relying on KNOWME semantics. For instance, energy consumed for a byte of data compression and transmission is independent of whether it is performed within the KNOWME framework or otherwise. In each subsection below we evaluate the energy cost of each component of KNOWME design using multiple available choices.

3.1 Impact of Software Platform on Energy Consumption

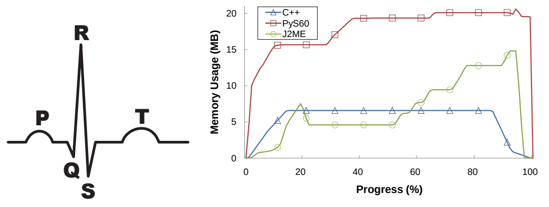
Selecting an appropriate software development platform is arguably the most important factor in any system design. The programming platform choice can determine the development cost in terms of person-hours as well as the system performance. Hence, in this section we first evaluate the design choices available for mobile phone software development in terms of their energy efficiency. There are three popular SDKs for programming Symbian Operating System based phones such as N95; namely, Symbian C++, Java 2 Micro Edition (J2ME), and S60 Python (PyS60), and show their relative energy efficiency.

We implemented a simple summation routine using all three programming platforms supported by N95. The application initializes a variable *sum* to zero and then does a summation in a loop with loop count. This is a trivial application that does not invoke any complex system calls or memory management routines. Hence, this kernel is a good estimator for the energy efficiency of the runtime environment. Table 1 shows the execution time. Note that qualitatively these results should come as no surprise; interpretive languages are slower than native execution. But quantitatively differences between various programming models is quite significant. PyS60 suffers severe execution overhead due to interpreter overhead. Prior comparison results [21] using Python on desktop environments showed that Python is only marginally slower than C since it uses Python modules to run natively on the hardware. However, PyS60 almost exclusively uses interpretation to dynamically generate native instructions. Hence, the overhead of PyS60 is extremely high, nearly 1000X worse than Symbian C++. J2ME does pay the penalty of managed environment. However, JVM on mobile phones uses several optimizations such as code caching to speedup execution, particularly in loops.

Loop Count	*C++*	*J2ME*	*PyS60*
10,000	<0.001s	<0.001s	0.069s
100,000	0.001s	0.030s	0.689s
100,000,000	0.612s	3.659s	688.379s

Table 1: Execution Time of Simple Summation on N95

In order to measure overheads when running more realistic applications we selected three functions that are of particular interest to WBANs, namely a heart beat detection algorithm (QRS Detection [19]), Advanced Encryption Standard (AES) encryption, GNU zip (Gzip) data compression. AES encryption is commonly used to transmit the data from the mobile phone to the back-end server for data protection. Gzip is also another



(a) (b)

Figure 3: (a) QRS (b) QRS Memory Usage

commonly used function to compress data before transmitting the data to the back-end server to reduce transmission costs. ECG signal is characterized using the P, QRS and T waves as shown in Figure 3(a). The QRS Detection Algorithm (QDA) detects R peak used as basis of reference in ECG segmentations and is necessary to recognize heart beat. We selected QDA implementation from the Open Source ECG Toolbox [30]. We ported this algorithm to each of the three programming languages.

The first 9 bars in the Figure 4 show the execution time of the three functions using the three programming platforms (the remaining bars will be discussed in the next section). Table 2, row labeled N95, shows the corresponding energy consumption in joules. In this paper, we used the Nokia Energy Profiler Tool [16] to report energy consumption values. The tool itself is fairly lightweight and adds negligible overhead. We used 10 minutes of ECG data, which is 180KB of data, as input to each of the three functions QDA, AES and Gzip. Based on results from Table 1 one would intuitively expect Python to have the lowest performance, and correspondingly the highest energy. Results from Figure 4 show that PyS60 performance on AES is just as bad (3 orders of magnitude) as shown in simple summation routine. However, for QDA PyS60 is about 40X slower than Symbian C++ (comparing Bars 1 and 3 in Figure 4) and has a corresponding 40X higher energy consumption (Column1 and Column 3 in Table 2). The improved performance of PyS60 in QDA case, compared to the 1000X performance loss seen in Table 1, can be attributed to efficient implementation of several built-in library functions, such as low-pass filtering in PyS60. For Gzip, PyS60 performed as fast as Symbian C++. Further analysis of PyS60 Gzip function showed that PyS60 implements Gzip as a native function written in Symbian C++ and included in PyS60 as an extension module. In essence, PyS60 implementation of Gzip is a Symbian C++ implementation of Gzip. To summarize, in all the tests, Symbian C++ shows the best performance.

3.1.1 Memory Consumption

To provide a better understanding of the observed execution time differences we provide how memory usage varies over time while running QDA across the three programming platforms. Memory consumption is a major source of execution time variations and hence energy consumption. As mentioned earlier, PyS60 and J2ME provide automatic memory management functions to reduce the burden on the user to deallocate memory correctly. Figure 3(b). shows the memory allocation and deallocation during QDA. The X-axis shows normalized time as a percentage of the total execution time of the corresponding programming model, and Y-axis shows the amount of memory occupied. In Symbian C++ implementation since the application programmer allocates and deallocated memory explicitly the size of memory allocated to the process does not continue to increase with time. On the other hand both PyS60 and J2ME experience increasing memory footprint over time. Furthermore, the baseline memory consumption in PyS60, even when the QDA is just beginning, is significantly higher than Symbian C++. While J2ME is more efficient during the initial execution phase it memory usage significantly deteriorates with execution time.

Note that while memory consumption provides good understanding of the observed execution time differences, there are also several micro-architectural interactions, such as cache misses, branch mispredictions that may lead to differing execution times. Since mobile phones do not provide performance counters it is difficult to measure these subtle micro-architectural interactions in our setup.

In summary, this section quantified the relative energy inefficiencies of three different software platforms. As mentioned earlier, it is qualitatively obvious that interpretive languages are likely to be slower than languages that run natively. But there is growing trend in traditional computing to use interpretive languages due to their programming simplicity and portability. However, in the WBAN domain given the stringent battery constraints native execution provides significant energy savings.

3.2 Sensitivity to Hardware Platform

One may raise the obvious concern, are the results from prior section are specific to the underlying hardware platform of N95? Do these application behave differently on a different mobile phone. In order to clarify these concerns, we repeated the same set of experiments on a Nokia E75 and Apple iPhone. Table 3 shows relevant hardware and software specifications where the mobile phones differ. All platforms are based on ARM cores but iPhone processor is nearly twice as fast as the other two and has also twice the amount of memory. Hence, memory related issues like garbage collection are likely to be more severe on Nokia platforms compared to iPhone. iPhone provides only one programming interface, namely iPhone SDK, which is closer to the Symbian C++ in terms of programming complexity.

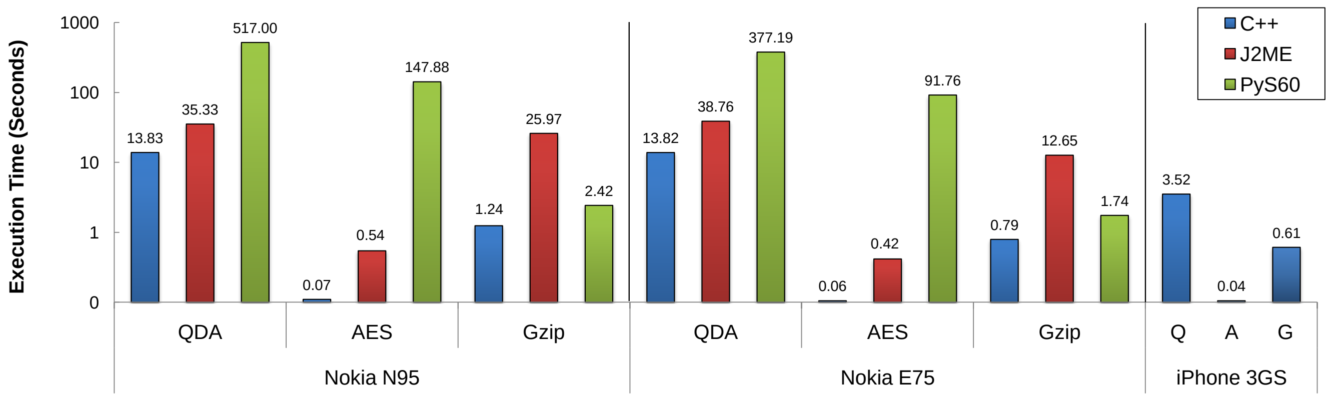


Figure 4: Execution Time of Three WBAN Functions on Three Programming Languages

Model	*QDA*			*AES*			*Gzip*		
	C++	J2ME	PyS60	C++	J2ME	PyS60	C++	J2ME	PyS60
N95	6.6J	19.34J	270.97J	0.03J	0.30J	77.62J	0.50J	12.22J	1.35J
E75	5.75J	15.95J	151.63J	0.02J	0.16J	36.89J	0.33J	4.93J	0.70J
iPhone 3GS	2.58J	n/a	n/a	0.04J	n/a	n/a	0.55J	n/a	n/a

Table 2: Energy Consumption For Three Processing Functions

The second set of 9 bars in the Figure 4 show the execution time of the three functions while running on E75. The last set of 3 bars in the Figure 4 show the execution time of the three functions while running on iPhone. Similarly in Table 2, rows labeled E75 and iPhone, show the corresponding energy consumption in joules. Comparing results between N95 and E75 the relative impact of programming language on execution time (and energy) across all applications remain the same. Also execution time decreases in almost all cases on E75. The primary reason for the improved performance and energy efficiency on E75 is that these applications are all single threaded and the higher frequency ARM 11 processor on E75 executes them faster. iPhone executes these applications even faster compared to Symbian C++ implementation on E75. Again, the reason is that iPhone processor is nearly 2X faster than E75. All these applications have similar code footprint, given that all platforms use ARM ISA. Hence, the relative execution time differences of the three application remains the same.

Due to resource limitations we restrict our evaluation in the rest of this paper to only N95 and show results only on this phone model. [1]

Spec	*N95*	*E75*	*iPhone 3GS*
CPU	Dual ARM 11 332Mhz	ARM 11 369HMz	ARM A8 600Mhz
Memory	128 MB RAM	85MB RAM	256MB
OS	Symbian S60	Symbian S60	iOS4

Table 3: Specification of the Mobile Phones

[1]If a reader is interested, we will consider putting the complete set results from E75 in an extended technical report.

3.3 Energy Consumption of the Sensors

After establishing the energy impact of software development choice, we now focus on the energy costs of sensing itself. In KNOWME built-in sensors such as ACC consume energy to perform the sensing operation while external sensors (ECG and OXI) cause mobile phone's battery drain due to Bluetooth communication. Table 4 shows the energy consumption of the built-in phone sensors (ACC) and the energy consumption when reading data from external sensors using Bluetooth. This data is obtained while sensing for a 10 minute interval using the sampling rates shown in the table. A Symbian C++ application is used for reading the sensor information. As mentioned earlier, ECG generates 300 sample per second while OXI generates 100 samples per second. However, these sensor samples are internally buffered in the sensor and bulk transmitted over the Bluetooth channel. Although ECG takes 300 samples it only transmits 4 samples per second, each packet has 75 sensor samples. OXI transmits 10 packets per second. Hence, the energy consumption while receiving data from OXI sensor is slightly higher than when communicating with ECG. Interestingly, when both ECG monitor and OXI concurrently send data the mobile phone is more energy efficient since the energy expenditure is much less than the sum of the energy spent when both sensors transmit data in isolation. We surmise that the energy expended in putting the bluetooth radio in an active listening mode is amortized over both sensors when hearing from the two sensors simultaneously. The Built-in ACC consumes 38 Joules for 10 minutes while generating 30 samples per second.

3.4 Energy Consumption of GPS

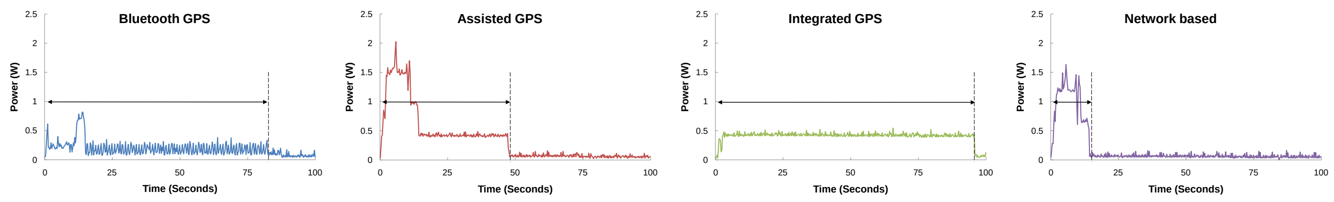


Figure 5: Energy Cost of Positioning Methods

Table 4 also shows the GPS energy consumption using assisted GPS technology used in N95. In the 10 minute interval we made two GPS readings and hence the sample rate is 0.0033 samples per second. GPS consumes significantly higher power per each sample than any of the other sensing functions in KNOWME. Given that GPS is an energy-intensive operation mobile phones provide multiple options for obtaining location information. In particular, when using N95 the user has four choices: Bluetooth based external GPS, Assisted GPS, Traditional GPS with no assist, and network-based GPS that uses only cell towers to provide approximate position information. Figure 5 shows energy consumption when GPS is processing a first request using all four approaches. Generally, once the position is known, the next GPS reading cost is reduced. For example, in the assisted mode GPS, the first reading cost for GPS is 29 Joules per sample, whereas from the second reading it is reduces to 25 Joules. The figure shows tradeoffs in the power consumption and the time to obtain the first GPS reading using all four methods. In Figure 5 the Bluetooth based external GPS power consumption shows only the mobile phone power consumption for establishing a Bluetooth link and read the GPS coordinates. There is a spike in the power consumption when the connection is established and the data is read over the Bluetooth channel and the power consumption drops down to the idle power when Bluetooth is ON but not actively sending/receiving data. The Assisted GPS curve shows a large power spike when it communicates with the cell tower to get the rough location first. Once the cell tower provides the orbital data of GPS satellites the GPS receiver on the mobile phone can narrow the search for satellite signals and quickly obtain the position information. In our measurements this approach took roughly 15 seconds to receive the position information from network and further 35 seconds to compute the precise position from a cold start. The third curve labeled Integrated GPS is the basic non assisted GPS. In this mode the GPS receiver continuously scans for the satellite information. This approach uses lower peak power but takes nearly 100 seconds before obtaining the position information due to the absence of any assistance from the cellular network; the total energy consumed by Integrated GPS is 33.9 Joules. The last curve shows the power consumption of Network based position information which, like Assisted GPS, communicates with cell tower to triangulate its approximate position information but no further communication with satellites is used. Hence network-based GPS also consumes roughly the same power as Assisted GPS initially when communicating with the cell tower for the 15 seconds. The power consumption then drops to idle power since it does not try to compute accurate position information by communicating with the satellites. Based on the total energy consumption (Power consumption * time to get GPS reading) our evaluations show Network-based GPS is best for saving energy in a WBAN (13 Joules per sample), even though location data is only approximate. Network-based GPS is more than 2X energy efficient than Assisted GPS. When precise location is needed then Assisted GPS is the best option.

Sensor	*Energy (J)*	*Sampling Rate*	*Xmit Rate*
Built-in ACC	37.804	30	30
ECG	114.846	300	4
OXI	137.433	100	10
ECG&OXI	156.419	300 & 100	4 & 10
Assisted GPS	53.994	0.003	0.003

Table 4: Energy Cost of Sensor Readings

3.5 Storage Costs

Once the sensor data is received the mobile phone may write the data to phone's flash memory for further analysis. Figure 6 shows the battery level of the mobile phone as we continuously sense data from ECG, OXI and write sensor data to the flash memory. The curve labeled ALL_UnBuffer shows the battery level on the mobile phone as we write each packet of data immediately to the local flash without any buffering. In other words, every sensor sample received is immediately written to the flash memory. It is well known that flash energy efficiency is significantly compromised for small writes. Flash writes must be done at the size of a page granularity, typically 4KB pages. If a smaller than page size write is performed usually the page that is being modified must be first read from the flash into a DRAM buffer and the bytes that are going to be written are updated in the DRAM buffer. Then the entire DRAM buffer is written back to flash. Hence, writes lead to a read-modify-write sequence in the Flash memory, where even a few bytes of write translate to a full page write, which is referred to as write amplification effect [2].

The curve labeled ALL_Buffer shows the battery level if we buffer the writes to DRAM and send large chunks to write to the flash. In this case we buffered sensor data till we receive at least 100 packets from a sensor. Then we write the buffered data to the flash. As can be seen buffering improves the battery life from 240 minutes to 299 minutes. Just a note of caution that the battery life time here does include the cost of using Bluetooth to receive data as well. Hence, the 240 minutes of battery life is not just due to flash writes only. Rather we focus on the difference in the battery life time with and without buffered writing to the flash, rather than the absolute values. The difference of 50 minutes translates into an additional 25% increase in the battery life time in this experiment.

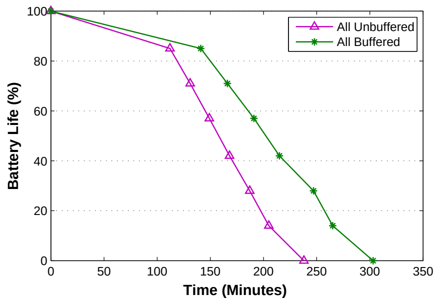


Figure 6: Battery Drain With Storage

3.6 Compute Versus Communicate

The mobile phone in KNOWME is not just a sensor data collection node but as described in Section 2 there is an Analyzer module that can perform local data analysis of the sensed data and a Transmitter module that transmits data to the back-end server, optionally using compression and-or encryption. Hence, the last trade-off we consider in this study is the energy consumption cost of performing local data analysis versus performing data analysis on the back-end server but pay for the data communication energy. This compute versus communication cost is not unique to KNOWME as we expect most WBANs to make this fundamental trade-off.

3.6.1 Communication Costs

To quantify the energy costs of communication we created a testbed. The testbed can send data from mobile phone to the back-end server using either: 3G, EDGE, Wi-Fi. AT&T broadband network is used for 3G and EDGE, while an 802.11g Linksys router is used for Wi-Fi. We varied the data size transmitted to the back-end from 100KB to 1000KB. As the performance of mobile wireless networks vary from place to place and from carrier to carrier, we tested all of data transfer measurements from the same location during early morning within one hour to reduce network congestion

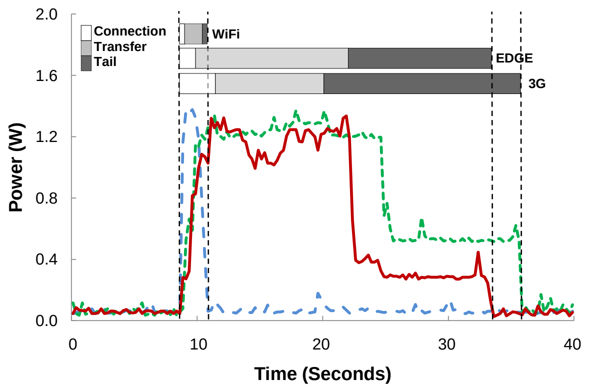


Figure 7: Three Phases of Transmissions

problems with other users. We repeated this study multiple times, but always done at the same time of the day, to measure day to day variations.

A data transmission consists of three phases, namely a connection phase, a transfer phase and a tail phase. Figure 7 shows the power consumption during the three phases for the three wireless data transmission approaches. In this experiment we used a 200KB data transfer. Due to shorter range, even though Wi-Fi has much higher bandwidth the peak power consumed is only slightly worse than 3G and EDGE radios on N95.

Figure 8 shows data transmission (uplink) energy costs of three of wireless interfaces as we increase the size of data transfer from 100KB to 1000KB. Wi-Fi is the most energy efficient across all data packet sizes. Hence, the overall energy consumption using Wi-Fi is significantly less than either 3G/EGDE in our setup. Obviously, due to limited Wi-Fi coverage a practical WBAN implementation will most likely use either 3G/EDGE for real time data transfer for mobile users. In this experiment, 3G consumes more energy than EDGE until about 400KB of data size. In order to understand the reason, Table 5 shows average connection and tail energy costs of all three network interfaces. The connection and tail energy of 3G are much higher than EDGE, while the transfer energy is lower. Hence for small data packets 3G consumes more energy than EDGE. As the data packet size increases beyond 400KB, the transfer energy dominates the overall energy costs and hence 3G consumes less energy than EDGE.

Meidum	*Connection Energy (J)*	*Tail Energy*
EDGE	0.346	2.987
3G	2.331	10.752
Wi-Fi	0.132	0.166

Table 5: Connection and Tail Energy Costs

3.6.2 Local or Remote Computation

In KNOWME the most complex data analysis function is to detect user state to identify long phases of physical inactivity (sedentary behavior). The choice of

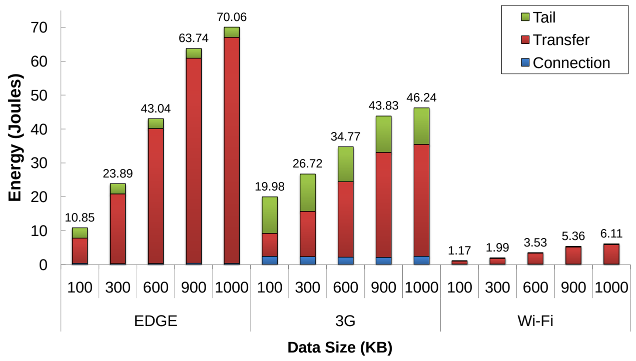


Figure 8: Data Transmission Costs (Uplink)

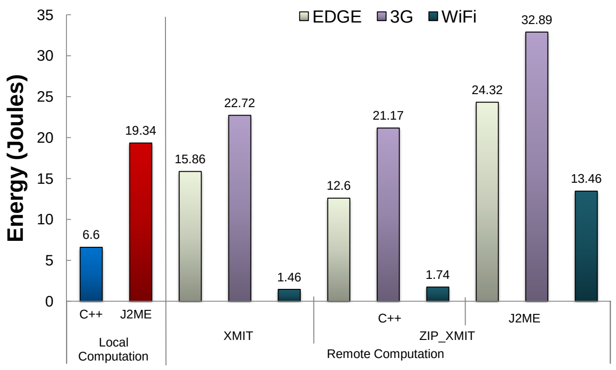
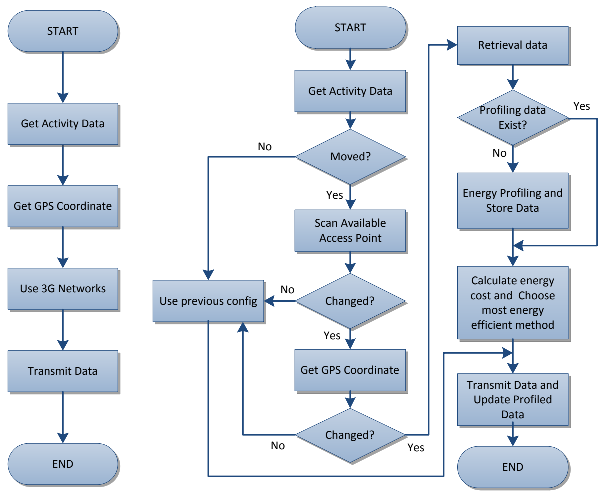


Figure 9: Local vs Remote Computation of QDA

whether to perform the user state detection on the mobile phone or on the back-end depends on the total energy cost taking several factors into account. Consider a simple case where we are interested in performing QDA on 10 minutes of ECG data, which is 180KB of data. Figure 9 shows the energy cost of this computation for local and remote computation. The first three bars in the graph show the energy cost of local computation when QDA is implemented in C++, J2ME or Python. The second set of three bars show the transmission cost to perform data analysis on the remote server using three different transfer options: EDGE, 3G and WiFi. The last set of 6 bars show the energy cost when data is compressed and then transmitted. The compression algorithm is implemented in C++ and J2ME and the transfer options are EDGE, 3G, WiFi.

Let us consider J2ME implementation of QDA. The local computation cost is 19.34 Joules. The remote computation cost (without Gzip) varies between 1.46 Joules using WiFi up to 22.72 Joules using 3G. On the other hand the remote computation cost with Gzip implemented in J2ME varies from 13.46 Joules using WiFI to 32.89 Joules using 3G. Now consider C++ implementation of the QDA. The local computation cost is 6.6 Joules. The remote computation cost without Gzip remain the same as before since there is no software platform dependence on transmission cost. But when Gzip is also implemented in C++ the energy cost varies between 1.74 Joules using WiFi up to 21.17 Joules using 3G. When WiFi is available it is clearly energy efficient to perform remote computation. But when the user is roaming, which will be a common case in WBAN operation, local computation is better than EDGE or 3G cost when QDA is implemented in C++. But if QDA implemented using J2ME then then remote computation using EDGE and without Gzip compression (15.86 Joules) is better.

Even in this simple scenario the choice of remote versus local computation is a complex function of which software platform the application is developed under, the wireless radio being used and whether or not data is compressed. We even simplified the discussion by removing the network signal quality issues that may alter the energy costs dynamically; as explained earlier, in our setup we used the network during the least congested time and from a location with the best signal quality. Through this simple experiment we demonstrate that there is no single statically best choice when it comes to trading off energy costs of computation with communication.



(a) Baseline (b) Profile Energy and Store Data

Figure 10: Comparing Energy Saving Method

4. ACTIVE ENERGY PROFILING METHOD

The previous section quantified the energy cost of each sub-component within a WBAN. From the above results it is clear that there is no single design choice that optimizes energy cost under all operating conditions. Some of the choices can be statically made at design time for optimizing energy efficiency. For instance, it may be worth the additional programming effort to use native execution than to use managed or interpreted environments that are more portable but consume more energy. Similarly, buffering data in the flash storage before transmission adds a small delay before the data is received at the back-end. Even for WBANs that need real-time data at the back-end the additional delay is not a major impediment given that buffering improves energy efficiency by at least 25%. However, there are many other choices of a WBAN whose energy consumption costs cannot be statically determined. For instance, different WBANs may have different computational demands depending on their target domain. Furthermore, computational demands may also vary with time. When KNOWME detects sedentary behavior it may run new algorithms that activate user-specific approaches to encourage physical activity. Similarly the energy cost of network signal quality, compression and encryption choices need to be dynamically computed.

In this section we present a dynamic approach to energy management, called the Active Energy Profiling (AEP) approach. We first present a general approach for implementing AEP in any WBAN. Later we present our specific implementation of AEP within the KNOWME implementation limitations. An idealized AEP uses a short profiling phase where it tests the various choices for WBAN operation. The profiling phase first decides how much sensor sampling is necessary for achieving the required user state detection accuracy. Optimal sensor sampling rate for user state detection may be computed using approaches described in [33]. It then runs the sensor analysis algorithms, that are specific to the WBAN implementation, locally on the mobile phone. It will use sensor samples collected over a short profile time window to perform local analysis. AEP then measures the energy consumption for local computation. AEP measures the communication cost of transmitting the same sensor data to a back-end where the analysis algorithms may run on a server. It then receives the results from data analysis from the server. AEP measures the energy consumption of the data transmission using both uncompressed and compressed data. Once the profiling phase is complete AEP then switches to a regular operating mode. During regular operation sensors are sampled at the optimal rate as calculated during profiling phase for achieving desired accuracy. If local computation is more energy efficient than remote computation then AEP selects that option for data analysis. If remote computation is more efficient AEP then selects the wireless radio with least energy consumption as measured during profiling phase based on network signal quality. Thus during regular operation energy consumption is same as the lowest energy option measured during profile phase. However, the regular operating mode may drift from the optimal operating mode over time. Hence, the regular operating mode is interrupted whenever there is change in the system state. A change can be triggered due to three reasons: (1) when the user moves to a different location, or (2) when an interesting event is detected during sensor data analysis, or (3) after a predefined time quantum has elapsed. AEP as described above is a simple and yet powerful approach to automatically optimize WBAN's energy consumption.

There are several design parameters that need to defined for AEP to work efficiently. A few key issues are listed below:

- **Profile Duration** WBAN implementer must decide on the duration of the profile phase. The profile phase must at least collect minimum amount of sensor data before activating analysis algorithms. Furthermore, if the computation and communication cost varies with the input size the profile phase must be able to predict the expected energy consumption for any data size using the profiled energy measurements from the sample data size.
- **State Change Detection:** The designer must also define the criteria under which the regular operating mode is interrupted for collecting new profile data. One criteria could be time based where the designer specifies the time bounds for toggling between profiling and regular operating mode. Other approaches may use change in sensor data volume to trigger a new profile phase.
- **Profile Overhead:** Since most system changes are repetitive it is possible to store the profiled data for a given system state and use the stored data rather than redo the full profiling. This approach reduces the energy cost of profiling as well as the latency to transition to regular operating mode.

4.1 AEP Implementation in KNOWME

The specific implementation of AEP within KNOWME is shown in the flowchart in Figure 10. In this implementation we use AEP to optimize on the two most energy consuming operations in KNOWME, namely GPS and data transmission. The flow chart on the left shows the sequence of steps in the baseline KNOWME. The system collects activity data by sampling the sensors. It buffers sensor samples for 10 minutes. KNOWME runs a SA once every minute that uses just ACC data to determine if a user is sedentary or not. It then gets position information using Network-based GPS once every 10 minutes. KNOWME then uploads the 10 minute geo-stamped sensor data to a back-end server using 3G. The flow chart on the right in Figure 10 shows AEP decision tree. The system still collects sensor data and buffers sensor samples for 10 minutes, just as in the baseline KNOWME. At the start of the WBAN operation AEP collects basic information regarding compression energy cost per bit and how it scales with data size. It also computes the typical compression ratios obtained for the data collected from the first few data samples. It stores the energy costs of compression and compression ratios in a local database. AEP uses SA to decide if user has moved. In other words, if the user state is classified as sedentary, AEP assumes user has not moved position. When user has not moved then it simply operates the WBAN using the current operational settings. In our current implementation the operational settings include (1) Whether or not to collect GPS information, (2) whether or not to do data compression, and (3) which wireless radio to use in the presence of multiple transmission options. If the user state is classified as not-sedentary then the system assumes that user has moved. If the user has moved the system does not immediately start profiling. Instead it notes that user has moved and waits until the next energy intensive operation, which in our case is GPS and data transmission. Before the beginning the energy intensive operation AEP then scans for WiFi access points (APs) to detect change in location based on detected APs. If APs have changed then the system requests for position information using GPS. If a WiFi AP is available it then probes its internal database to see if there is any existing profile information for that WiFi AP and GPS position. If such a profile information exists it then uses that profile information to set the WBAN operational settings. If no such profile information exists then it runs a profile phase with 10 minute sensor data to measure the energy consumption with various wireless radios and data packet sizes with and without compression. Note that when multiple WiFi APs are available AEP profiles the energy cost for sending data from each of the WiFi APs. It then selects the most energy efficient setting from the profile run and sets the WBAN's operational settings. These settings are also stored in the profile information database and that entry is associated with a key formed by combining WiFi AP and GPS.

Using the above algorithm energy savings come from multiple optimizations: First, AEP can skip GPS sensing whenever user has not moved within a 10 minute interval. Second, AEP selects between 3G and WiFi AP. It can also select the least energy AP when multiple APs are available. The use of WiFi also significantly reduces the data transmission cost. Rather than running a new user state algorithm it simply uses KNOWME's SA to infer movement. Few limitations are worth noting. First, the sampling rate for external sensors in KNOWME are currently not programmable. Hence, we can not optimize external sensor sampling rate in current KNOWME. Second, KNOWME runs SA on ACC data for providing real time feedback to the user regarding their sedentary state. SA consumes little energy for doing the classification. Hence, SA computation is always run locally. The more complex user state detection using multi-modal signal processing is done on the back-end. The last limitation is that Symbian S60 does not allow user level access for selecting EDGE or 3G network. Hence, we could not select EDGE network even when EDGE uses less energy in cases where the data packet size is small. We always ended with 3G in our operating locations.

4.2 Results from AEP Implementation

For the data presented in this section we ran KNOWME on two N95 phones (identical in all respects, including battery age). On one phone we ran the baseline KNOWME and the second phone ran AEP implementation of KNOWME, as described in the previous section. In this experiment the user is moving throughout the experiment, detected as non sedentary by SA, between different locations and comes in contact with WiFi AP as well. The sensor layer consisted of two external sensors, ECG and ACC sensors. Figure 11 shows how the power consumption varies with time with both approaches for a 40 minute KNOWME run where data is transmitted to back-end once every 10 minutes. The figure is divided into three segments. The first segment (left most segment) shows how power consumption varies with time in the baseline KNOWME. The first spike in the power consumption is the GPS and second spike is 3G upload. The small spikes that occur once every minute correspond to the power consumption of SA that runs every minute. This pattern repeats every 10 minutes irrespective of user state.

The last two segments of Figure 11 show the power consumption with AEP. The second segment shows when the user is roaming and hence there is only 3G network. The last segment shows power consumption when the user comes in proximity of two known WiFi APs. The system first does a WiFi scan to see if user has moved. But at the start it has no position information. So it gets position information by GPS sensing just as in baseline KNOWME. After GPS sensing the new power spike corresponds to energy profiling. During this profile run the system tries to upload data using 3G with and without Gzip. Once the profiling phase is over the system uses 3G to upload compressed data as it was determined to be the best energy saving option. At the next interval there is a short WiFi scan power spike since the user is not sedentary. In this case the user has not moved to a new location and hence WiFi scan also determines the same since no APs are detected. During this intervals there is a clear absence of GPS and profiling power spikes since the system detects that the user has stayed within the same area as during the profile phase. Hence AEP simply uses WBAN settings that were selected from the previous profile phase. Finally at time 25 the user has moved to a new location where there are two WiFi APs in addition to 3G. At 30 before starting energy intensive GPS and data transmission operation AEP starts the profile phase. There is a power spike corresponding to WiFi scan which now determines that user has moved and it also determines (with a 5 second delay) that there are two new APs at

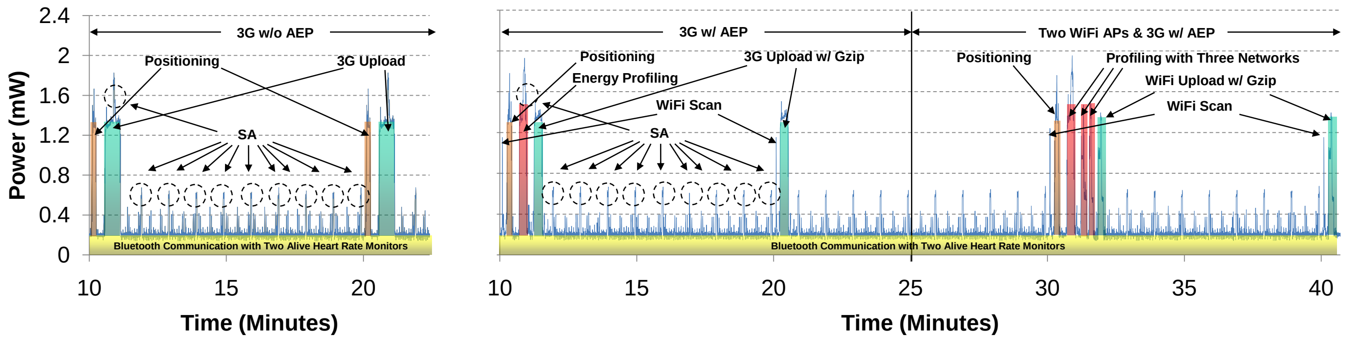


Figure 11: Power usage of various cases

this location. The system then uses GPS sensing which confirms that the user has in fact moved to a new location. The profile phase now has three spikes that correspond to the system sending data using the two WiFi APs and 3G. Once profiling is complete it selects WiFi. It continues to use WiFi access point to reduce overall energy consumption as long the user has not moved too far away. Note that this figure just demonstrates how AEP dynamically adapts to changing conditions. In a true WBAN operation the user does not move as frequently as we have shown in this figure. Hence, profiling is not done as frequently and the system operates in optimal mode for long periods of time before needing a new profile run.

The net energy reduction after one hour of operation with AEP is shown in Figure 12. Each bar measures the total energy for one hour of KNOWME operation. The first bar is the baseline KNOWME. The second bar is AEP with only 3G. The third bar is AEP with 3G and two WiFi APs. The bluetooth data transmission costs stay the same in all approaches, as to be expected given that external sensor sampling rate could not be programmed, due to hardware limitation in our current off-the-shelf components implementation. The cost of profiling is less than 1% of the total energy cost. However, the benefits of profiling are clear. AEP with just 3G reduces the energy consumption from 1114 Joules to 980 Joules, a 12% improvement. AEP with two WiFis and 3G reduce the energy cost to 918 Joules, an 18% energy reduction. Note that if we remove bluetooth energy costs that we could not alter with AEP the gains from AEP are even more impressive. Out of the 314 Joules (1114 - 800 Joules for Bluetooh) AEP targets for reduction it reduces the energy consumption to 118 joules, a 62% energy improvement.

5. RELATED WORKS

The popularity of WBANs for health monitoring has been increasing in the recent years. They are being deployed to assist in physical rehabilitation [11], obesity monitoring [3, 6], assisted living [17]. All these prior studies focused on the usability of the system in terms of computer-human interface. But as shown in our research, understanding energy implications of WBAN, from design to operation, will significantly improve the battery life and benefit many of these prior WBAN implementations.

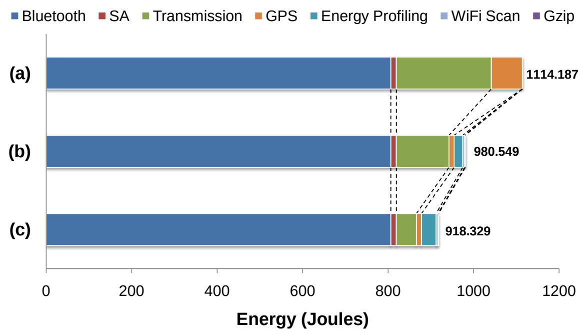


Figure 12: Comparison of Energy Consumption (a) Baseline (b) AEP with 3G Only (c) AEP with Two Wi-Fi APs & 3G

In [8] the authors described MAUI which is an automated system that can switch between local and remote computation based on round trip time of a request response cycle. MAUI relies on flexibility of managed code environment to create two versions of any computational task, one that runs locally on a phone and second that runs remotely on a server. It estimates the amount of task state that needs to be transferred between local and remote sites. During runtime it computes the energy cost of local computation versus transferring state to a remote site based on current network conditions. It then invokes remote computation using RPC whenever that option is more energy efficient. Our research differs in two notable ways. We first characterizes the energy consumption of *all aspects of a typical WBAN*, software development platform, sensing, GPS, data buffering, and finally remote versus local computation. Furthermore, the AEP approach takes a broader set of criteria into consideration to automatically decide on the best WBAN operational point. For instance, it can avoid GPS sensing when user is sedentary. It measures the energy cost of data transmission by profiling across various network interfaces, including multiple APs. It can also avoid profiling by simply storing profiled data in a database indexed by GPS and AP as the key. We believe our AEP implementation can benefit from using MAUI framework to automatically split any task into remote and local methods. One note of caution is that MAUI extensively relies on managed runtime capabilities. But from an energy efficiency view point, our results show that native methods such as Symbian C++ are typically more efficient than J2ME.

Viredaz *et al.* [31] discussed methods for improving energy consumption in hand-held devices. They detect periods of idle time in a mobile device and use voltage frequency scaling to reduce power consumption. Shih *et al.* [24] showed that using wake-on-wireless a PDA can be put into sleep mode and woken up only on an incoming call or when the user is actively using the device. Turdecken [27] demonstrates how to use hierarchical power management to reduce energy consumption. In this context they attach a low power mote to a mobile node and use the mote to continuously monitor incoming packets. The mote wakes up the mobile device when an incoming packet is detected. The notion of hierarchical energy management is exploited at a much finer granularity in EEMSS [34]. EEMSS categorizes all the sensors on a mobile phone into a hierarchy and then activates low energy sensor which will in turn decide when to activate a higher energy intensive sensor.

User state detection is one of the fundamental tasks of a WBAN, such as KNOWME [3] and UbiFIT [6]. These studies perform feature extraction on the sensor signals and then run classification algorithms on the extracted features. The tradeoff between state detection accuracy and sensor sampling rates are explored in [32, 12, 13]. Wang *et al.* [33] developed a framework for selecting optimal sensor sampling for achieving highest state detection accuracy when energy is constrained. In SeeMon [12] sensor energy is conserved by providing feedback between end user application requirements and the sensor layer. In [1] Aghera treat sensing, communication and computation as tasks and use a task assignment algorithm to reduce the energy consumption of the mobile phone. Constandache *et al.* [7] show that GPS power consumption can be reduced by predicting user movement patterns rather than trying to compute location using GPS continuously. In [18] authors proposed RAPS that estimates user velocity and uses GPS only when the user has moved sufficiently far to reduce GPS energy consumption. We believe KNOWME will further benefit from user movement prediction to reduce sensor energy, but currently we do not have use any prediction algorithms.

Energy efficiency in the context of communication is well studied. Energy efficient algorithms for wireless sensor networks have been proposed in [29, 25, 26]. Recently studies have also done energy measurement of wireless data transmission using various network interfaces [5]. They show that data transmission energy varies widely from one location to another and may also vary at the same location depending on the time. Hence, they also argue that dynamic routing is necessary for optimizing energy. In [4] the authors introduce Wiffler which is designed for a vehicular network for optimizing data throughput. Wiffler can change network interface between WiFi and 3G depending on network condition. It uses historical data to predict future available WiFi APs. Since Wiffler is geared toward vehicular network it cares more about bandwidth, whereas a WBAN is sensitive to battery consumption. Breadcrumbs [15] tracks user's movement to generate connectivity forecasts Ra *et al.* [22] focused on dynamically selecting between various wireless radios. They introduce SALSA that uses Lyapunov optimization framework to automatically decide when to send data and when to defer data transmission and wait for better channel availability so as to optimize the overall energy-delay tradeoffs. Again many of these approaches use prediction of WiFi APs or user movement to optimize energy of at least one component that is used in KNOWME. We need to carefully develop these prediction models in KNOWME context and we can then take advantage of these prior studies to further reduce energy. Even without relying on user movement prediction AEP provides significant benefits.

It has been shown that the size of ECG data can be reduced by using special compressing methods using an autoregressive feature. Some of them use lossy compression method to get high compression ratio [10]. In particular, Fahim *et al.* [9] shows a three phase encoding-compression-encryption mechanism of which higher compression ratio is up to 20.06 on mobile phones. Although the compression methods can reduce storage as well as transmission energy cost, we need to consider the energy cost of the compression method itself. In section 3.6.2, we show a case where the energy costs of data compression is higher than the transmission cost.

While there are disparate sources of some of the energy consumption information none of the previous studies have done a systematic and comprehensive analysis of the energy consumption of a WBAN starting with the initial system design choices to the data transmission. We believe that our research also sheds new light on issues such as programmability and energy efficiency, energy compression costs of sensor data and the energy cost of storing data locally on mobile phones. Given the energy consumption uncertainty in WBAN it is necessary to use a profile based dynamic adaptation to minimize energy consumption across all layers of a WBAN.

6. CONCLUSIONS

Limited battery life of mobile phones is a significant bottleneck to a wider deployment of WBANs. As the popularity of WBANs continue to increase there is a need to understand where exactly the energy consumption goes in a WBAN. This paper presents a comprehensive quantification and analysis of energy consumption of the various components in KNOWME, which is one implementation of a WBAN. There are several tradeoffs a designer has to make to reduce energy consumption. At the software level our results showed that improving energy efficiency is as important a metric as programmer productivity in WBANs. We also quantified how energy consumption can be curtailed by reducing the sensor sampling rate or using approximate sensing, such as network based location sensing. We also show that while the quantitative results presented here may differ from one mobile phone to another the qualitative conclusions drawn are more broadly valid. We should also note that the findings of this work may also be helpful for other classes of applications that use sense-compute-communicate cycle.

Inspired by the results from our measurement study we implemented Active Energy Profiling (AEP). AEP is a dynamic approach that automatically optimize the energy consumption based on the real time operating conditions of a WBAN. Given the uncertainty of energy consumption across various WBAN tasks no single statically selected operating point can effectively reduce WBAN energy consumption. Hence, AEP uses a short profile run to dynamically compute the energy costs of various WBAN tasks and automatically selects the operating point that best reduces energy consumption during that time interval. In a one hour operation of KNOWME AEP can reduce the energy consumption by 12% even when there are no alternatives to 3G data transmission. When multiple network interfaces are available AEP can reduce the overall KNOWME energy by 18%.

7. REFERENCES

[1] P. Aghera, T. Rosing, D. Fang, and K. Patrick. Poster abstract: Energy management in wireless healthcare systems. In *Information Processing in Sensor Networks, 2009. IPSN 2009. International Conference on*, pages 363 –364, april 2009.

[2] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for ssd performance. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70, 2008.

[3] M. Annavaram, N. Medvidovic, U. Mitra, S. Narayanan, G. Sukhatme, Z. Meng, S. Qiu, R. Kumar, G. Thatte, and D. Spruijt-Metz. Multimodal sensing for pediatric obesity applications. In *Proceedings of International Workshop on Urban, Community, and Social Applications of Networked Sensing Systems (UrbanSense08)*, November 2008.

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

[5] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293, 2009.

[6] S. Consolvo, D. W. McDonald, T. Toscos, M. Y. Chen, J. Froehlich, B. Harrison, P. Klasnja, A. LaMarca, L. LeGrand, R. Libby, I. Smith, and J. A. Landay. Activity sensing in the wild: a field trial of ubifit garden. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1797–1806, 2008.

[7] I. Const, S. Gaonkar, M. Sayler, R. R. Choudhury, and O. Cox. Energy-efficient localization via personal mobility profiling. In *Proceedings of The First Annual International Conference on Mobile Computing, Applications, and Services - MobiCASE 2009.*

[8] 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 *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 49–62, 2010.

[9] S. Fahim and K. Ibrahim. Enforcing secured ecg transmission for realtime telemonitoring: A joint encoding, compression, encryption mechanism. *Security and Communication Networks, John Wiley & Sons, Ltd*, 1(5):389–405, Aug 2008.

[10] E. B. L. Filho, N. M. M. Rodrigues, E. a. B. da Silva, M. B. de Carvalho, S. M. M. de Faria, and V. M. M. da Silva. On ecg signal compression with 1-d multiscale recurrent patterns allied to preprocessing techniques. *IEEE transactions on bio-medical engineering*, 56(3):896–900, 2009.

[11] E. Jovanov, A. Milenkovic, C. Otto, and P. de Groen. A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation. *Journal of NeuroEngineering and Rehabilitation*, 2(1):6, 2005.

[12] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song. Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 267–280, 2008.

[13] A. Krause, M. Ihmig, E. Rankin, D. Leong, S. Gupta, D. Siewiorek, A. Smailagic, M. Deisher, and U. Sengupta. Trading off prediction accuracy and power consumption for context-aware wearable computing. In *ISWC '05: Proceedings of the Ninth IEEE International Symposium on Wearable Computers*, pages 20–26, 2005.

[14] M. Li, V. Rozgica, G. Thatte, S. Lee, A. Emken, M. Annavaram, U. Mitra, D. Spruijt-Metz, and S. Narayanan.

[15] A. J. Nicholson and B. D. Noble. Breadcrumbs: forecasting mobile connectivity. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, MobiCom '08, pages 46–57, 2008.

[16] Nokia Energy Profiler v1.2. `http://www.forum.nokia.com/Technology_Topics/Application_Quality/Power_Management/`.

[17] N. Noury. Ailisa : experimental platforms to evaluate remote care and assistive technologies in gerontology. In *Enterprise networking and Computing in Healthcare Industry, 2005. HEALTHCOM 2005. Proceedings of 7th International Workshop on*, pages 67 – 72, june 2005.

[18] J. Paek, J. Kim, and R. Govindan. Energy-efficient rate-adaptive gps-based positioning for smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 299–314, 2010.

[19] J. Pan and W. J. Tompkins. A real-time qrs detection algorithm. *Biomedical Engineering, IEEE Transactions on*, BME-32(3):230–236, March 1985.

[20] K. Patrick, W. G. Griswold, F. Raab, and S. S. Intille. Health and the mobile phone. *American Journal of Preventive Medicine*, 35(2):177 – 181, 2008.

[21] L. Prechelt. An empirical comparison of seven programming languages. *Computer*, 33(10):23–29, 2000.

[22] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely. Energy-delay tradeoffs in smartphone applications. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 255–270, 2010.

[23] J. Ryder, B. Longstaff, S. Reddy, and D. Estrin. Ambulation: A tool for monitoring mobility patterns over time using mobile phones. volume 4, pages 927 –931, aug. 2009.

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

[25] M. Shin, P. Tsang, D. Kotz, and C. Cornelius. Deamon: energy-efficient sensor monitoring. In *SECON'09: Proceedings of the 6th Annual IEEE communications society conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 565–573, 2009.

[26] A. Silberstein, R. Braynard, and J. Yang. Constraint chaining: on energy-efficient continuous monitoring in sensor networks. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 157–168, 2006.

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

[28] J. C. Sriram, M. Shin, T. Choudhury, and D. Kotz. Activity-aware ecg-based patient authentication for remote health monitoring. In *ICMI-MLMI '09: Proceedings of the 2009 international conference on Multimodal interfaces*, pages 297–304, 2009.

[29] L. Stabellini and J. Zander. Energy efficient detection of intermittent interference in wireless sensor networks. *In International Journal of Sensor Networks (IJSNet)*, 8(1), 2010.

[30] The Open Source ECG Toolbox (OSET). `http://ee.sharif.edu/~ecg`.

[31] M. A. Viredaz, L. S. Brakmo, and W. R. Hamburgen. Energy management on handheld devices. *Queue*, 1(7):44–52, 2003.

[32] Y. Wang, B. Krishnamachari, Q. Zhao, and M. Annavaram. Markov-optimal sensing policy for user state estimation in mobile devices. In *Proceedings of The First Annual International Conference on Mobile Computing, Applications, and Services - MobiCASE 2009*, 2009.

[33] Y. Wang, B. Krishnamachari, Q. Zhao, and M. Annavaram. Markov-optimal sensing policy for user state estimation in mobile devices. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 268–278, 2010.

[34] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *MobiSys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 179–192, 2009.