Decomposing power measurements for mobile devices

Andrew Rice
Computer Laboratory
University of Cambridge
Cambridge, United Kingdom
Email: acr31@cam.ac.uk

Simon Hay
Computer Laboratory
University of Cambridge
Cambridge, United Kingdom
Email: sjeh3@cam.ac.uk

***Abstract*—Modern mobile phones are an appealing platform for pervasive computing applications. However, the complexity of these devices makes it difficult for developers to understand the power consumption of their applications. Our measurement framework is the first we have seen which can produce fine-grained, annotated traces of a phone's power consumption and is designed to develop an understanding of how particular aspects of an application drive energy use. We are using our framework to analyse the power consumption of Android-based G1 and Magic handsets and show that particular choices of message size and send buffer can alter the energy required to send data by an order of magnitude in certain cases.**

***Keywords*-energy measurement; mobile communication; power measurement; wireless LAN**

I. Introduction

The mobile phone is the world's most popular computing platform. Over 2 billion people now own a mobile and in established markets it is not uncommon for individuals to own more than one. In developing markets (such as Africa) the mobile phone is commonly the only accessible technological device serving as a clock, a torch and a calculator as well. For example, in 2007 there were 25 mobile phones per 100 people in Uganda compared with 1.6 computers or 15.6 radios [1]. Furthermore, modern smart phones are a rapidly growing segment of the market providing markedly more capability than conventional handsets. These devices contain myriad communication interfaces, significant processing power and storage and numerous sensors ranging from simple light sensors to GPS tracking.

These trends have made the mobile phone a particularly appealing platform for pervasive computing applications, but as with all battery-powered devices, controlling and managing power consumption is an issue. These applications have a number of notable characteristics with respect to power consumption. Firstly, many context-sensitive applications will run continually in the background collecting sensor information or waiting for a trigger condition and so even a small power requirement has the potential to impact the device more heavily than other power hungry but short-lived programs. Secondly, many applications rely on the participation of a large group of users to be useful. Perhaps one of the most compelling cases is the concept of participatory sensing [2], where a large number of volunteers can use their smart phones to gather sensor data in the background. This can subsequently be used for all sorts of purposes, such as to build up a picture of environmental factors [3] or generate collaborative maps [4]. For these applications to succeed the cost of participation must be small compared with the direct benefit gained [5] and so the power impact of running the application must be minimised. Finally, these applications often operate in varying conditions but with flexibility about how a particular task is performed — for example, sensor readings can be reported immediately or stored up for later transmission. Applications should choose the best approach from the various options available and dynamically integrate with the overall usage of the device.

In this paper we examine the power consumption of two Android 1.5-based handsets and show in detail the energy used by particular actions taken by an application. Decomposing the energy consumption of the phone into independent parts is a considerable challenge and so we present our approach to data collection and analysis of traces. We explore the energy costs of using different wireless networks to demonstrate the need for the high fidelity and update rate offered by our technique, which is applicable to any programmable mobile device.

II. Power measurement and interpretation

Our power measurement system delivers a number of key features:

- **Automated test execution** The tests proceed (as much as permitted by the device) without requiring user interaction. This removes a major source of variability and allows the tester to increase confidence in a result by repeatedly executing the same sequence of actions to eliminate random noise (but not systematic error).
- **Batch operation** A whole sequence of tests can be run without intervention. Test scripts written in our purpose-designed simple language are interpreted dynamically without requiring any changes to the software running on the phone.
- **Untethered operation** No physical connections (except those to the battery itself) are required to any of the interfaces on the phone and there are no networking

978-1-4244-5328-3/09/$25.00 ©2009 IEEE

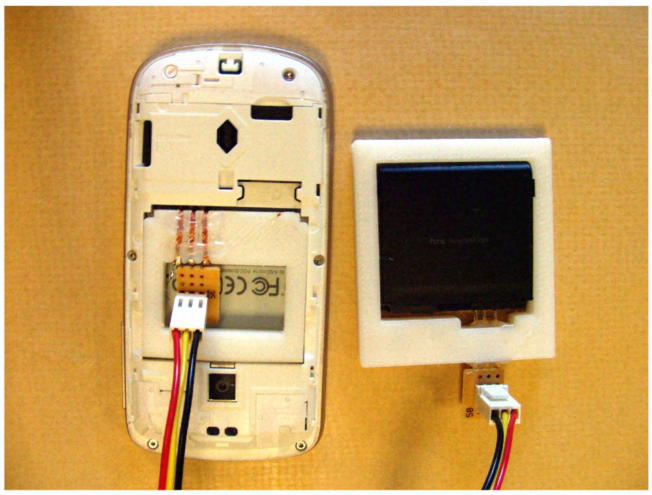

Figure 1. Replacement battery and battery holder for Magic handset.

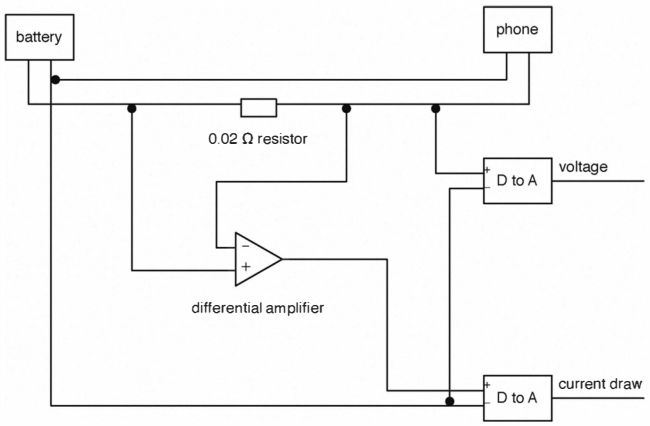

Figure 2. Block diagram of power monitoring apparatus.

requirements beyond the initial download of the test script and final upload of the results. This means, for example, that tests can easily be run examining the costs of changing network or using an external device.

- **No hardware modification** It is not necessary to modify the test device at all other than to construct a replacement battery and install a standard application. Neither permanent hardware changes nor 'root' software access are required, making the technique accessible to normal developers.

The measurement system is centrally orchestrated by the Power Server. This is responsible for sending test scripts to the phone and collecting and aligning the various traces and log files. A single client program is run on the phone itself which is responsible for acquiring a test script and executing the required actions. The client program collects a timing log of these events which is uploaded to the Power Server at the end of the test.

III. Measurement hardware

The phone's power consumption is measured by inserting a high-precision 0.02 Ω measurement resistor in series between a battery terminal and its connector on the phone. We do this by replacing the battery of the handset with a printed plastic replacement[1] (Figure 1). We produced replacement batteries and battery holders to fit both the G1[2] and Magic[3] handsets.

We use a National Instruments PCI-MIO-16E-4 sampling board to measure the voltage across the phone battery and also the voltage drop across our measurement resistor (which we first amplify with an instrumentation amplifier) at 250 kHz. Inserting the measurement resistor increases the circuit resistance, and therefore its power consumption. This is not a problem for our purposes as this is typically less than 1% of the total power. The circuit schematic is shown in Figure 2. We also developed a prototype microcontroller-based board suitable for sampling on the move but at a lower rate than the National Instruments device, which we hope to use in future experiments.

The presence of high-frequency components within the phone's electronics does not cause exceptionally rapid changes in the power consumption. This is most likely to be due to buffering within the phone caused by capacitance in the circuit or voltage regulation. We inspected a number of traces using a high-speed (1 GHz) storage oscilloscope in order to satisfy ourselves that our sampling rate was sufficient to capture all features of the trace.

The expected power consumption of a resistor is trivially calculated and so we use a selection of high-precision resistors to calibrate our device. Figure 3 shows on the y axis our measured power draw and on the x axis the power calculated using Ohm's Law. The necessary scale factor was then computed using a linear regression through the resulting data points, with a RMS of residuals of 0.0001 (4dp).

IV. Test client

The test client running on the handset process proceeds as in Figure 4. The phone handset first connects to the Power Server and downloads a test script. It then enters the preparation phase and stabilises its power consumption. A predetermined sequences of actions is performed to create a synchronisation pulse. This is used in the data analysis phase to correlate the timing log from the phone with the recorded data. The phone next executes the test script recording the time at which each action is performed. Once the script is complete the power consumption is stabilised once more and a final synchronisation pulse is emitted before uploading the timing log back to the Power Server.

[1]We produced our replacements using a Reprap 3D printer — http://reprap.org/ — and will make the designs available

[2]http://www.t-mobileg1.com/

[3]http://www.htc.com/www/product/magic/overview.html

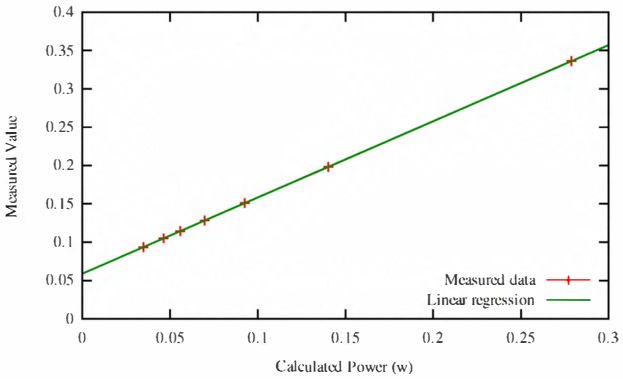

Figure 3. Results of calibration with known resistors.

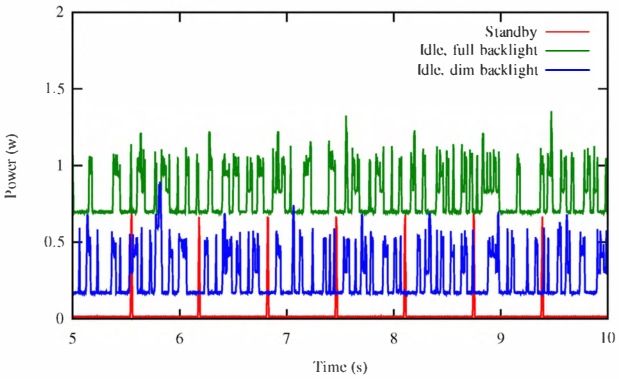

Figure 5. Energy consumption of the G1 when idle.

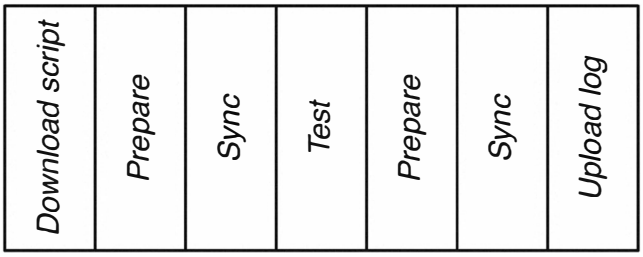

Figure 4. Block diagram of a trace showing the execution steps of a test.

A. Stabilising the power trace

Our primary interest is in breaking down the overall power consumption of the phone into its constituent parts. Ideally one would identify components which are causing variation in the trace, characterise their consumption and then switch them off. For components such as the CPU this is not an option because the operating system is preemptively multitasking and so other processes are intermittently waking up and consuming resources: Figure 5 shows the variation in power when the phone is ostensibly idle. Instead, we run a low-priority background process in a busy-loop. This consumes all spare CPU cycles and contributes greatly to stabilising the power trace. A small uncertainty is introduced by this technique because we cannot distinguish between CPU load created by the test and the background load. However, for the purposes of understanding the peripheral hardware in the phone (such as the networking devices) this should have little effect.

B. Trace synchronisation

Many of the features in the energy trace last only a fraction of a second. For example, a scan for available wireless networks lasts around 500 ms, while the transmission of a single packet takes only a few milliseconds. For this reason it is important to align precisely the times recorded for different events on the phone with the samples recorded on the measurement PC. This alignment allows us to annotate the power trace at each instant with the action taking place on the phone.

We achieve this by embedding a synchronisation pulse *inside the phone's energy trace* by switching the backlight of the phone on and off in a predetermined pattern. Switching the backlight of the phone from off to full brightness increases the power consumption of the device by more than half a watt over a period of a few milliseconds (Figure 5). We exploit this effect to embed two easily-recognisable 32-bit Gold codes [6] at either end of the trace, with on representing a 1 and off a 0. The pulse sequence is shown in Figure 6.

At the end of the test the timing log recorded by the device contains the switching times for each edge in the synchronisation pulses. We combine these times with estimated values for the power consumption in the two backlight states to generate a hypothesised sequence representing our expectation of the power trace for the pulse. It is valid to assume in this case that the relative drift of the two clocks over the synchronisation time period will be negligible.

To find the sample which corresponds to the start of the synchronisation pulse we compute a *cross-correlation function* of the power trace and the hypothesised signal by incrementally increasing the offset of the latter and computing the sum of the squares of the difference between the signals (also shown in Figure 6). This function drops to zero at the point where the two signals line up; the properties of the Gold codes ensure it is very unlikely that they will match in the wrong position. In all our tests a visual inspection has shown this to be a robust way to determine the sample number of the start of both pulses.

The latest versions of the Android operating system fade the backlight on and off, resulting in diagonal lines in the synchronisation pulses on the power trace rather than sharp edges. If we attempt to match the square hypothesised signal against this it lines up incorrectly, with the vertical edges half way along the sloping lines where we should have the rising edge at the base of the upward sloping line and the falling edge at the top of the downward one (Figure 7(a)). To

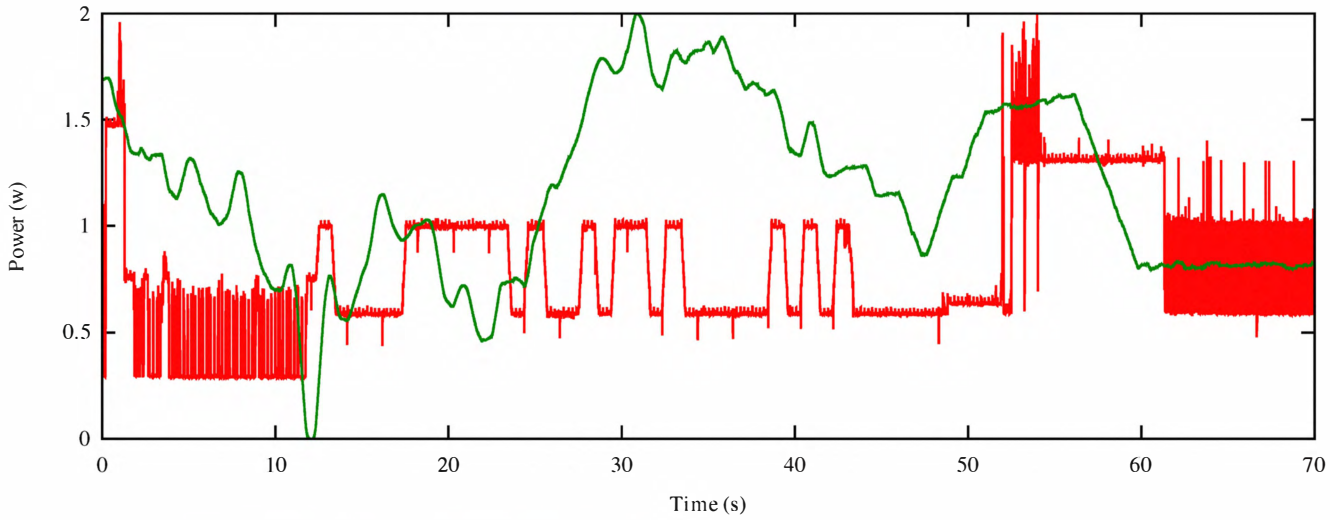

Figure 6. A synchronisation pulse (in red, approximately between seconds 12 and 42) and the result of the cross-correlation function with the hypothesised signal (in green).

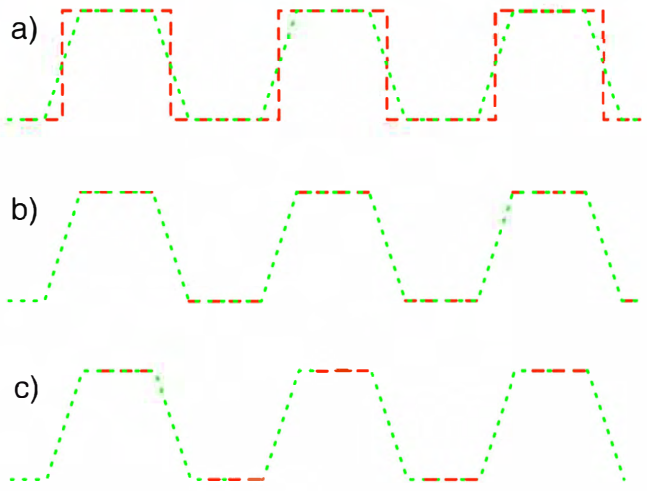

Figure 7. Fading the backlight causes a square pulse to misalign. The green dotted line represents the actual trace, while the red dashed line represents the hypothesised signal in each case.

counteract this, we remove the edges from the hypothesised signal and leave gaps where the slopes will be (Figure 7(b)). Changing the backlight setting is an inter-process call on the phone handset and so there is sometimes a slight delay between making the API call and the change taking effect. We therefore also leave the start of each hypothesised pulse section blank (Figure 7(c)). This means that multiple places along the trace are good matches, so we choose the latest maxima.

This calibration procedure gives us four values: s_1 and s_2 correspond to the sample number for the start of the first and second pulse, and t_1 and t_2 which correspond to the time that the first and second pulse were begun. From this we compute

$$r = \frac{s_2 - s_1}{t_2 - t_1} \quad \text{and} \quad o = t_1 - s_1/r \tag{1}$$

where r is the sample rate between the two pulses and o is the time offset between the start of the power log and the start of the timing log. The sample s which corresponds to some time t is therefore calculated as

$$s = r(t - o) \tag{2}$$

Once the alignment has been calculated in this way, new estimates are formed for the power consumption with the backlight on and off and the calibration process is re-run with a new hypothesised signal based on these estimates to ensure as accurate a result as possible regardless of the physical device. One measure of the accuracy of our synchronisation is to look at the variance in r across our test runs. In a perfect world this would have a value of 4000 ns — we were never out by more than 3 ns.

C. Baseline calibration

Many of the tests involve switching on some component of the phone, waiting for it to initialise and then examining the additional energy costs of using the device. We support this process by embedding baseline power calibration in our test scripts. The test writer first annotates the script to indicate that a particular set of steps should be used as calibration information. When processing the log files we compute the average power consumption of these steps and remove it from subsequent steps. As an example, this allows us to dissociate the energy cost of transmitting data over WiFi from the ongoing power requirement to keep the WiFi interface active.

```
NONE:1:ToggleWakeLock:true                    # Force the device to remain awake
NONE:1:ToggleTelephony:false                  # Disable the cellular radio
NONE:1:ToggleWifi:false                       # Disable the WiFi radio
NONE:1:WaitTelephonyDisconnect
NONE:1:WaitWifiDisconnect
NONE:1:ToggleCPU:true                         # Start a background busy thread
PRESYNC:1:SetBacklight:1                      # First synchronisation pulse
NONE:1:DoSleep:800
PRESYNC:1:SetBacklight:0.1
...                                           # Remainder of sync pulse omitted
BASELINE:1:DoSleep:5000                       # Calibrate baseline power
MEASURE:1:ToggleWifi:true                     # Enable the wireless network
MEASURE_CONT:1:WaitWifiConnect                # Wait for a connection
NONE:1:DoSleep:5000                           # Wait for 5 seconds
BASELINE:1:DoSleep:10000                      # Calibrate Wifi idle power
MEASURE:1:OpenSocket:192.168.0.210:8060:1     # Open a TCP connection
NONE:1:DoSleep:5000                           # Wait for 5 seconds
MEASURE:25:SendTCP:25:1448:false              # Send 1448 bytes over TCP
MEASURE_CONT:1:DoSleep:3000                   # Wait for 3 seconds
NONE:1:CloseSocket                            # Close the TCP connection
...                                           # Start of sync pulse omitted
POSTSYNC:1:SetBacklight:0.1                   # Trailing synchronisation pulse
NONE:1:DoSleep:800
NONE:1:ToggleCPU:false                        # Release CPU
```

Figure 8. Parts of an example test script

The example script in Figure 8 measures the power consumption for switching on the Wireless LAN, holding it on for baseline calibration and sending 1448 bytes of data.

V. Network traffic monitoring

It is useful to observe the network traffic alongside the power trace of the mobile device in order to analyse the costs of different methods of sending and receiving data. We connected the PC recording the power trace to a wireless access point and configured it to run a DHCP server to emulate a typical network the phone might join. We then called libpcap[4] from within the Power Server application to record all packets seen on that interface.

The framework combines the synchronisation information embedded in the power trace with the timing log from the phone and the network traffic information in order to generated annotated graphs of power consumption. An example of this output is shown in Figure 9 (which we discuss in the following section).

VI. Power consumption analysis

Pervasive computing is a vision of communicating devices and so understanding energy costs of this communication is of great importance to application developers. We now present our study of the power consumed by sending data over a wireless network. The interaction between different layers in the hardware and software stack creates considerable differences in energy consumption. We believe that this provides significant motivation for measurement frameworks such as ours.

All the data presented in this section was gathered using the Magic handset, but results for the G1 were almost identical.

A. Connecting to the network

Figure 9 shows part of the energy trace of connecting to a WiFi network and obtaining an IP address using DHCP, annotated using the method described above with each IP packet sent and received. Without the packet labels this trace would be relatively hard to interpret, but the aligned annotations show clearly the costs of each aspect of the connection process. Note that the repeated DNS requests come from the operating system itself and are for a Google server; Android attempts to make contact at regular intervals, and these communications also show up in other test runs, distorting the results. The annotation allows them to be identified and accounted for. The actions taken by the phone when connecting to the network are prescribed by the various Internet RFCs. For example, the ARP Probe packets are designed to discover if there is another host on the network already using the phone's desired IP address [7]. The number of probe packets and the delay between them forms a significant fraction of the connection time (and energy).

B. Idle power

Figure 10 shows excerpts of the energy trace of a handset when connected to only the 3G, 2G or WiFi networks. Interestingly, in this case WiFi actually has the lowest idle power cost, followed by 3G and then 2G. Although the spikes are more frequent (every 100ms, corresponding to receiving base station beacons), the base power is lower than maintaining a connection to the cellular network.

It is not possible to draw concrete conclusions from these measurements because of the many factors which we have yet to investigate. The locations of the various base stations will have an effect on the power consumed by the radio and the building itself will have different attenuation properties at the different radio frequencies involved.

However, these measurements do demonstrate that one cannot always assume that one particular networking technology will have the lowest power consumption. For example, 2G networking is provided as an option in the phone's interface to reduce power consumption—in this particular case it is the highest power option.

C. Data transmission

One might intuitively expect that the energy cost of sending each byte of data reduces, or at worst remains constant, as the total amount of data sent increases. However, this is not the case. Figure 11 shows the number of joules required to transmit each kilobyte of data for increasing total message size. The baseline calculation functionality in our framework was used to remove the residual costs of running the phone and so these numbers are the actual amount of additional energy required to send the data. The graph shows the result of 10 test repetitions run at each 1 kilobyte interval.

Part of the reason for the noise in this data is due to the fact that other processes on the phone are also using the network. In this test case they are attempting DNS lookups of particular Google servers. The evidence of this activity is visible in the packet trace collected by the Power Server.

[4] http://www.tcpdump.org/

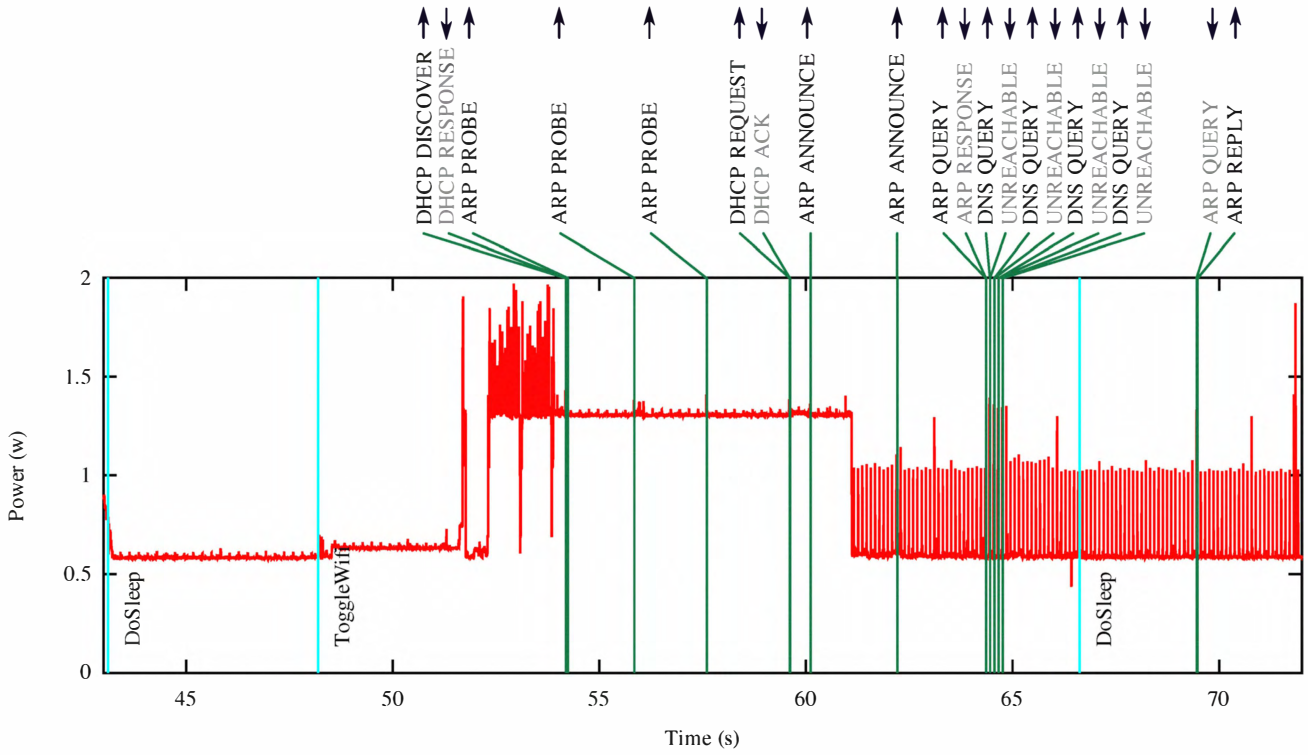

Figure 9. An extract from the energy trace of connecting to a WiFi network and obtaining an IP address, annotated with each IP packet sent and received.

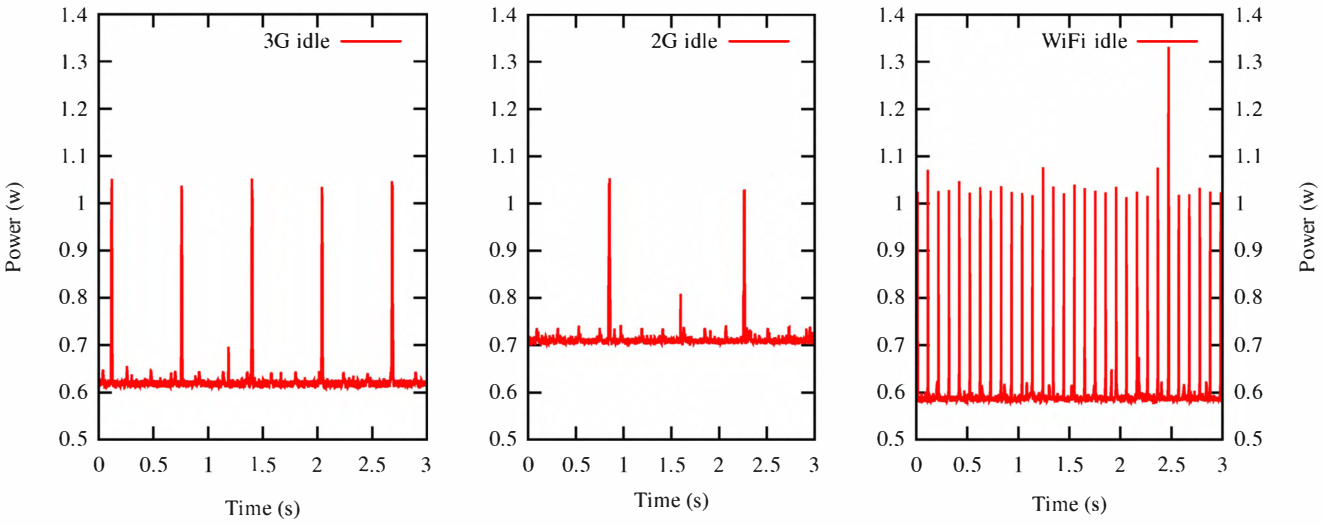

Figure 10. Idle power when connected to 3G, 2G and WiFi networks.

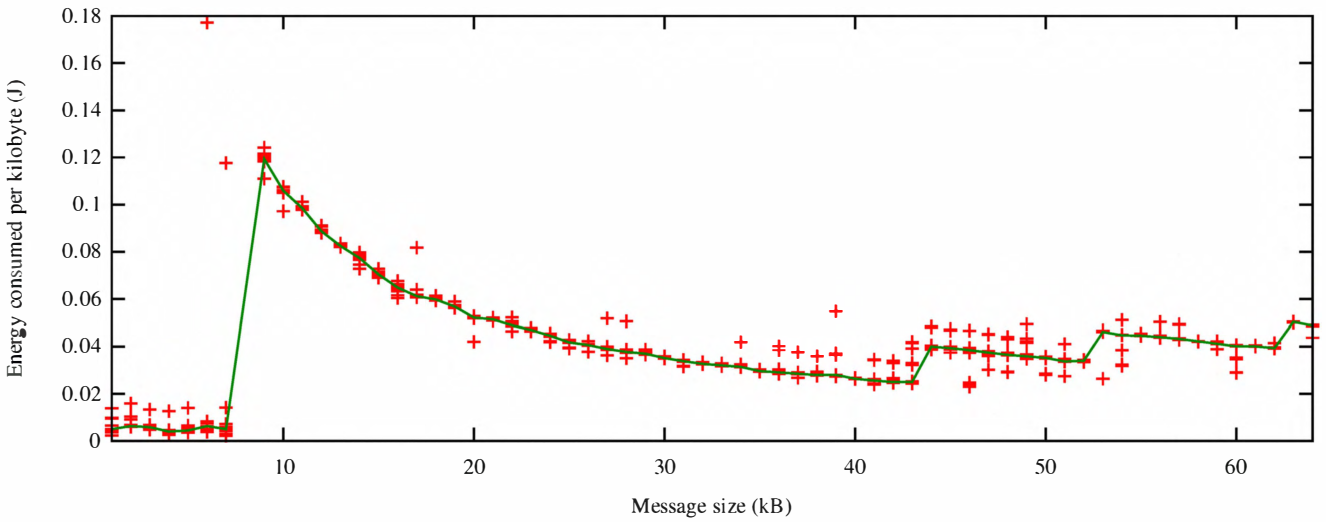

Figure 11. Variation in energy cost per unit data with total message size.

In future versions we hope to add filtering to discard results compromised by other activity taking place on the phone.

There is a clear jump in the cost per byte for 7 KB of data compared with 8 KB. Figure 12 shows these two instances in more detail. When sending 8 KB or more of data (Figure 12(right)) there is a considerable period of high energy consumption after the last packet has been sent which is not present when sending a 7 KB message (Figure 12(left)).

It is not clear why this is occurring and we can find no explanation in any of the relevant networking standards. Use of a separate wireless card and the Wireshark[5] packet sniffer demonstrated that there is no activity on the wireless network for this period. However, regardless of whether this is due to crossing some power management threshold or simply a bug in the wireless firmware, it has a significant impact on the energy requirements of sending a message. A pervasive sensing application on one of these devices would minimise power by batching a data update into chunks of around 7 KB but incur a significantly larger cost by batching to 8 KB.

D. Send buffer size

As a final example we consider the impact of the size of the send buffer used by the application developer. Android applications are written in the Java programming language and network data is sent by getting the OutputStream object associated with a Socket instance. Data is then sent over the network by calling the write method on the socket and passing an array of bytes to send.

The operating system interprets this as a request from the application to send the entire byte array immediately to the client socket without waiting for further data. The size of this array therefore causes significant changes in energy costs. Figure 13 shows how the energy cost per KB varies with changes in the size of the byte array passed from the application for a message of 1 KB and a message of 32 KB. Note that the buffer size (on the horizontal axis) is shown on a log scale. For both of these messages the choice of buffer size can cause a tenfold difference in the energy cost!

We expect the energy cost per KB for the 32 KB message to be lower, since the fixed costs are amortised over more data. However, thee 1 KB line collapses from its flat trend to the same level as the 32 KB line beyond a certain buffer size. This is because of the 7f/8 KB barrier we saw in the previous section: the change in cost is due to the number of frames sent over the Ethernet regardless of how much data is in them, and with a large enough send buffer there are sufficiently few frames that the 1 KB message does not cross this barrier.

E. Summary

Developers will often view choosing the amount of data to collect into a single message as a smooth trade-off between latency and energy consumption. Our data shows that for our handsets this is not the case. Similarly, the choice of send buffer size is often a secondary performance consideration whereas we have shown that it can cause an order of magnitude increase in the energy cost of a message. The annotations produced by our Power Server have been vital in drawing these conclusions: they make it clear when in the trace particular events are occurring and they have allowed us to run a large number of detailed tests by permitting automated measurement of elements in the power trace.

VII. Related Work

Dutta et al. present an ingeniously simple design for energy metering *in situ* by augmenting switching regulators [8],

[5]http://www.wireshark.org/

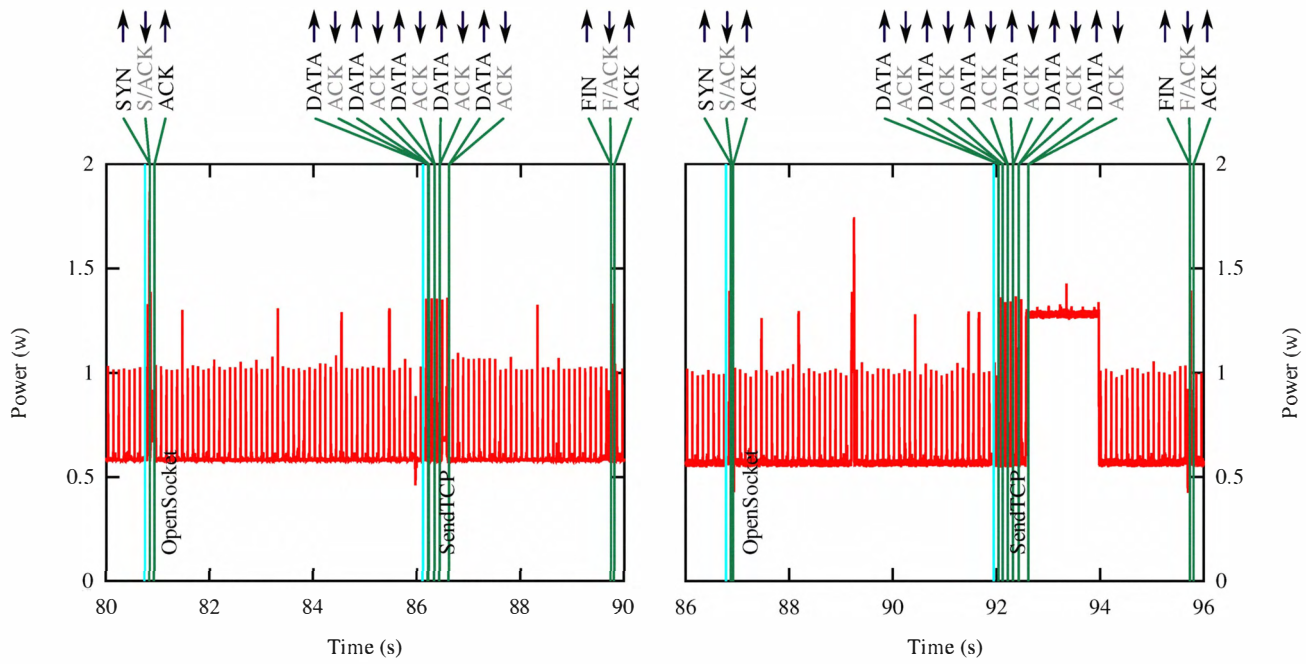

Figure 12. Extracts from the energy traces of sending 7 KB (left) and 8 KB (right) of data over WiFi

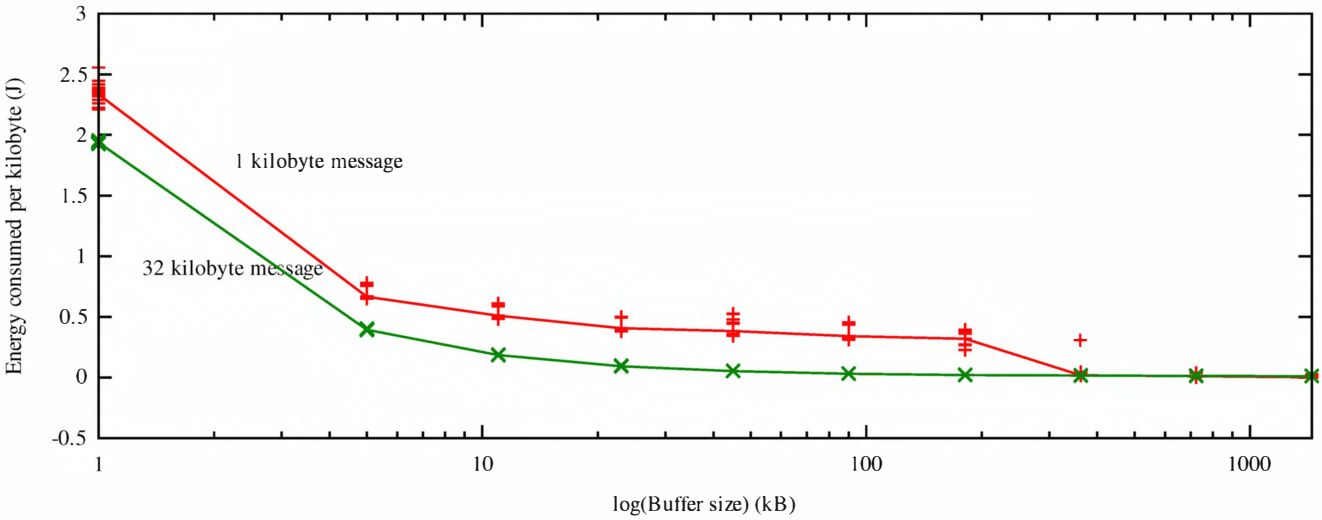

Figure 13. Variation in energy cost per unit data with buffer size.

and Fonseca et al. build on top of this hardware platform to apportion energy costs of components in embedded network devices to individual activities [9]. This allows developers to quantify the effects of different approaches, but requires significant hardware and operating system modification.

Flinn et al. have also contributed a significant body of work in the area of measuring and reducing the power consumption of larger mobile devices, including quantifying the energy consumption of a pocket computer [10] and the PowerScope tool for profiling energy usage [11].

Significant efforts have been made to reduce the energy consumption of wireless communication; while some system for measuring the power draw is required to evaluate these mechanisms, these have generally operated at a fairly coarse level. Pering et al. measured the voltage and current at the network interface cards in a similar manner to our proposal, but sampled only every 10 ms and did not attempt to align the trace with specific actions [12]; similarly, Mohan et al. looked at the overall power required by the sensors for their pervasive application but did not investigate any further [13].

There has been less work on in-depth energy monitoring on mobile phones. Platform manufacturers have offered some tools and guidance, but users have traditionally only cared to know the total energy remaining. As a result, most systems, such as Nokia's useful Energy Profiler[6], tend to have low resolution with slow response times. Google also presented useful advice on reducing energy usage on Android phones supported by basic power measurements. [7]

VIII. Conclusions and Future Work

Power consumption is often quoted on a coarse scale. This provides typical values and is useful information for users of the devices. However, application developers can benefit from knowing the detail behind these measurements and we have shown that seemingly small decisions can have a significant impact on power consumption. We have described our measurement framework and how it can be used to create a fine-grained understanding of energy consumption. A particular advantage of our system is the ability to annotate the power measurements with phone and network activity. We have shown how the synchronisation information required can be embedded in the measurement trace itself.

We have investigated the cost of sending messages over a wireless network. This is a particularly common feature of pervasive computing applications. Our results have shown interesting phenomena. Nevertheless, the search space for potential savings remains huge. We propose to investigate networking further, particularly on the move, and to study location providers and sample rates but we welcome suggestions of other interesting aspects.

Finally, it is inevitable that different devices (and different firmware) and different operating scenarios will display different behaviour. For example, we found that in our testing location the idle power consumption of the 2G network is higher than that of the 3G and wireless networks. Therefore it is informative to consider how device or platform developers might extend their hardware and APIs in future to integrate fine-grained power measurement and make 'third party' solutions like ours unnecessary. We have shown that simple metering must be augmented with additional contextual information to produce an explanation for a particular trace.

IX. Acknowledgements

We would like to thank Brian Jones, Alastair Beresford and Rob Harle for their advice and guidance and Andy Hopper for his insight and support.

References

[1] International Telecommunications Union, "World telecommunication/ICT indicators database 2008," *Technical report*.

[2] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory sensing," *WSW '06 at SenSys '06*.

[3] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda, "PEIR, the personal environmental impact report, as a platform for participatory sensing systems research," *Mobisys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services*, Jun 2009.

[4] J. J. Davies, A. Beresford, and A. Hopper, "Scalable, distributed, real-time map generation," *Pervasive Computing, IEEE*, vol. 5, no. 4, pp. 47 – 54, Oct 2006.

[5] K. Mansley, A. R. Beresford, and D. Scott, "The carrot approach: Encouraging use of location systems," *UbiComp '04: Proceedings of the Sixth International Conference on Ubiquitous Computing*.

[6] R. Gold, "Optimal binary sequences for spread spectrum multiplexing (corresp.)," *Information Theory, IEEE Transactions on*, vol. 13, no. 4, pp. 619 – 621, Jan 1967.

[7] D. Plummer, "Ethernet Address Resolution Protocol," RFC 826 (Standard), Internet Engineering Task Force, Nov. 1982.

[8] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler, "Energy metering for free: Augmenting switching regulators for real-time monitoring," *ISPN '08: Proceedings of the Seventh International Conference on Information Processing in Sensor Networks*.

[9] R. Fonseca, P. Dutta, P. Levis, and I. Stoica, "Quanto: Tracking energy in networked embedded systems," *OSDI '08: Proceedings of the 8th USENIX Symposium on Operating System Designs and Implementations*.

[10] K. Farkas, J. Flinn, G. Back, D. Grunwald, and J. Anderson, "Quantifying the energy consumption of a pocket computer and a Java virtual machine," *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, Jun 2000.

[11] J. Flinn and M. Satyanarayanan, "Powerscope: a tool for profiling the energy usage of mobile applications," *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pp. 2 – 10, Jan 1999.

[12] T. Pering, Y. Agarwal, R. Gupta, and R. Want, "CoolSpots: reducing the power consumption of wireless mobile devices with multiple radio interfaces," *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*, Jun 2006.

[13] P. Mohan, V. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, Nov 2008.

[6] http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/Plug-ins/Enablers/Nokia_Energy_Profiler/

[7] http://code.google.com/events/io/sessions/CodingLifeBatteryLife.html