ARTICLE IN PRESS

Pervasive and Mobile Computing ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Measuring mobile phone energy consumption for 802.11 wireless networking

Andrew Rice, Simon Hay *

Computer Laboratory, University of Cambridge, Cambridge, United Kingdom

ARTICLE INFO

Article history:
Received 10 April 2010
Received in revised form 19 July 2010
Accepted 27 July 2010
Available online xxxx

Keywords:
Energy measurement
Mobile communication
Power measurement
Wireless LAN

ABSTRACT

The complexity of modern mobile phones makes it difficult for developers to understand the power consumption of their applications. Our measurement framework produces fine-grained, annotated traces of a phone's power consumption which we are using to develop an understanding of how particular aspects of an application drive energy use. We ran a large number of automated tests using Google Android G1, Magic, Hero and Nexus handsets and present results for the average energy consumption of connection and data transmission over 802.11 wireless networks. Our results show that the optimal choice of data transmission strategy is different between handsets, operating systems, and device context.

1. 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.

* Corresponding author.
E-mail addresses: acr31@cam.ac.uk (A. Rice), sjeh3@cam.ac.uk (S. Hay).

1574-1192/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.pmcj.2010.07.005

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

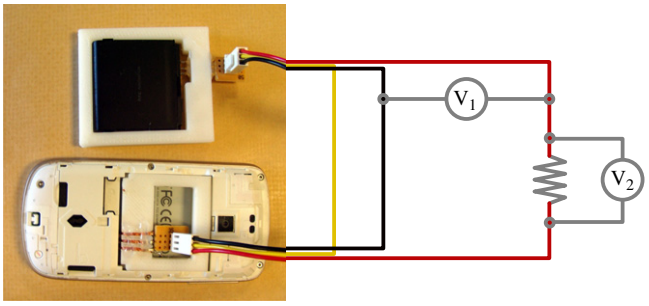

Fig. 1. Replacement battery and battery holder for the magic handset.

In this paper, we first present our measurement platform which allows us to make large numbers of measurements of particular operations on the handset in an automated fashion. We go on to examine the power consumption of Android-based handsets and show that changes to the DHCP process have reduced the energy consumed when connecting to a wireless network from approximately 6 to 1.5 J. We also report the energy costs of sending data over the wireless link and show that the choice of message size has a significant impact on energy costs and that this relationship shows different trends on different handsets. Our results are just a few examples of the fact that the strategy for optimising energy consumption change between handsets, operating systems, and the device context.

2. 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 the 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 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 the 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.

3. 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] (Fig. 1). We produced replacement batteries and battery holders to fit the G1,[2] Magic,[3] Hero[4] and Nexus 1[5] handsets. In all of the tests that follow the G1 and Magic handsets (running Android version 1.1) produced indistinguishable results. The results for the Hero handset are for Android 1.5 and the Nexus are for Android 2.1.

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 measurement points are shown in Fig. 1 from which simple application of Ohm's law yields Power $\propto V_1.V_2$.

[1] We produced our replacements using a Reprap 3D printer (http://reprap.org/).
[2] http://www.t-mobileg1.com/.
[3] http://www.htc.com/www/product/magic/overview.html.
[4] http://www.htc.com/www/product/hero/overview.html.
[5] http://www.google.com/phone.

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

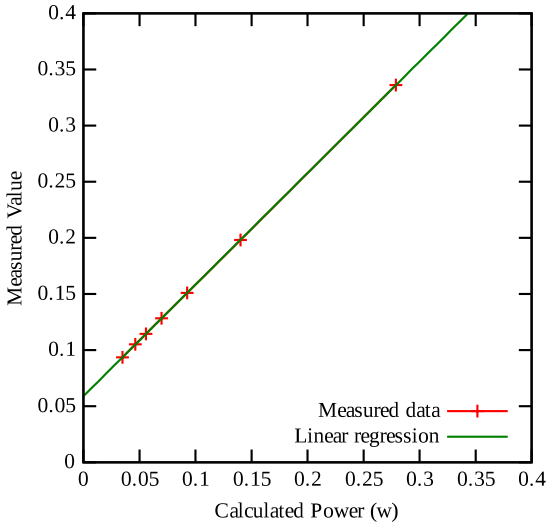

Fig. 2. Calibration with known resistance.

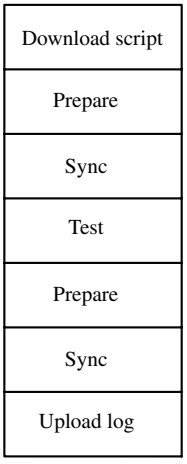

Fig. 3. Execution stages.

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. Fig. 2 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 an RMS of residuals of 0.0001 (4 dp).

Fig. 4 shows an example trace when switching on a G1 phone. The annotations of each boot phase and average power measurements were made manually.

4. Test client

The test client running on the handset process proceeds as in Fig. 3. 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 sequence 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.

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

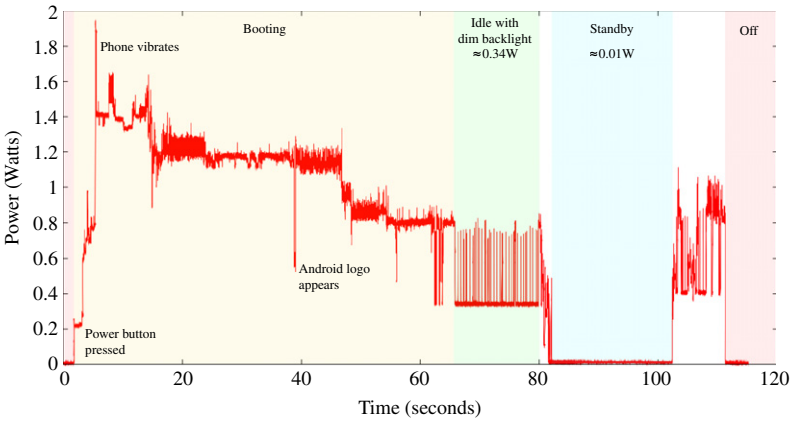

Fig. 4. Instantaneous power consumption when switching on a G1 phone.

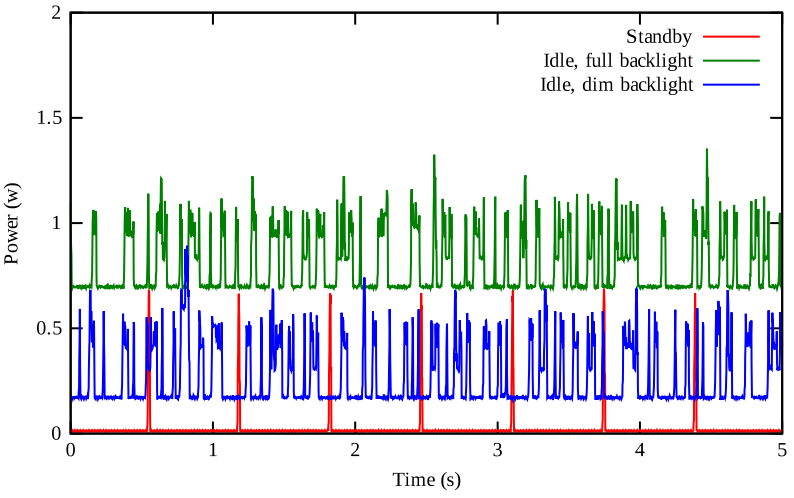

Fig. 5. Energy consumption of the G1 when idle.

4.1. 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: Fig. 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.

4.2. 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 (Fig. 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 Fig. 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

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

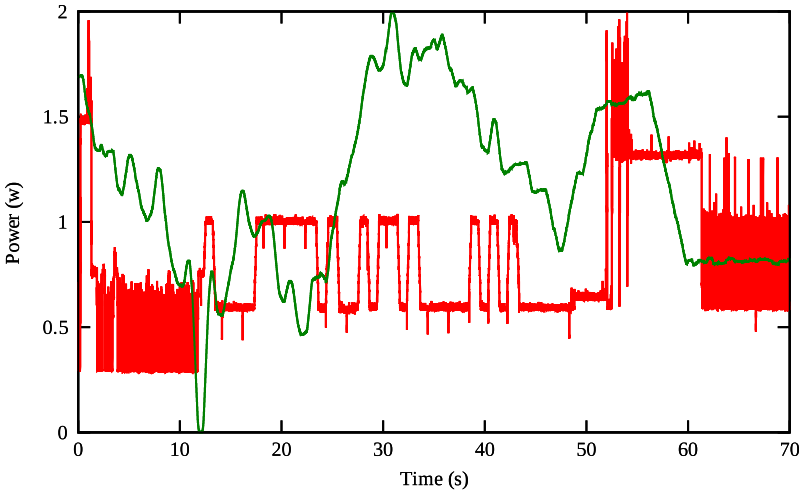

Fig. 6. A synchronisation pulse (in red, approximately between seconds 12 and 42) and the result of the cross-correlation function with the hypothesised signal (in green). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

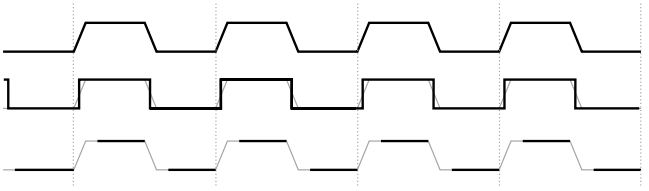

Fig. 7. A partial hypothesis trace (bottom) is necessary because a square hypothesis (middle) will misalign on the true trace (top).

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 Fig. 6). This function drops to zero at the point where the two signals line up; the properties of the Gold codes ensure that 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.

Recent 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 (Fig. 7(middle)). To counteract this, we remove the edges from the hypothesised signal and leave gaps where the slopes will be. 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 (Fig. 7(bottom)). These blank "don't care" sections mean that multiple points along the trace are good matches, so we choose the latest maximum.

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 and our results are never out by more than 3 ns.

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

```
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 25Kb TCP using 1448 byte buffer
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
```

Fig. 8. Parts of an example test script.

4.3. 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.

The example script in Fig. 8 measures the power consumption for switching on the Wireless LAN, holding it on for baseline calibration and sending 25 kB of data using a 1448 byte buffer. Each line of the script corresponds to a step in the test and contains a number of fields separated by the ':' character. The first field indicates the type of action to be taken by the measurement framework. The most relevant of these are the BASELINE action which informs the measurement system to calculate a baseline power consumption using the average power consumption for the duration of the step. The MEASURE action causes the power consumption (minus the current baseline power measurement) of the step to be recorded. The average power consumption (Watts) and the total energy consumed (Joules) are calculated. The number in the second field indicates the number of units of activity present in the step which is used to produce a normalised, unit-cost, energy consumption. In the example, there are 25 units in the SendTCP step sending 25 kB of data. This causes the system to automatically calculate the cost of sending 1 kB of data and add it to the logs. The MEASURE_CONT action indicates to the system that it should treat this step as a continuation of the previous one and make a measurement over the entire duration of both.

5. 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[6] 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 generate annotated graphs of power consumption. An example of this output is shown in Fig. 9 (which we discuss in the following section).

6. 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

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

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

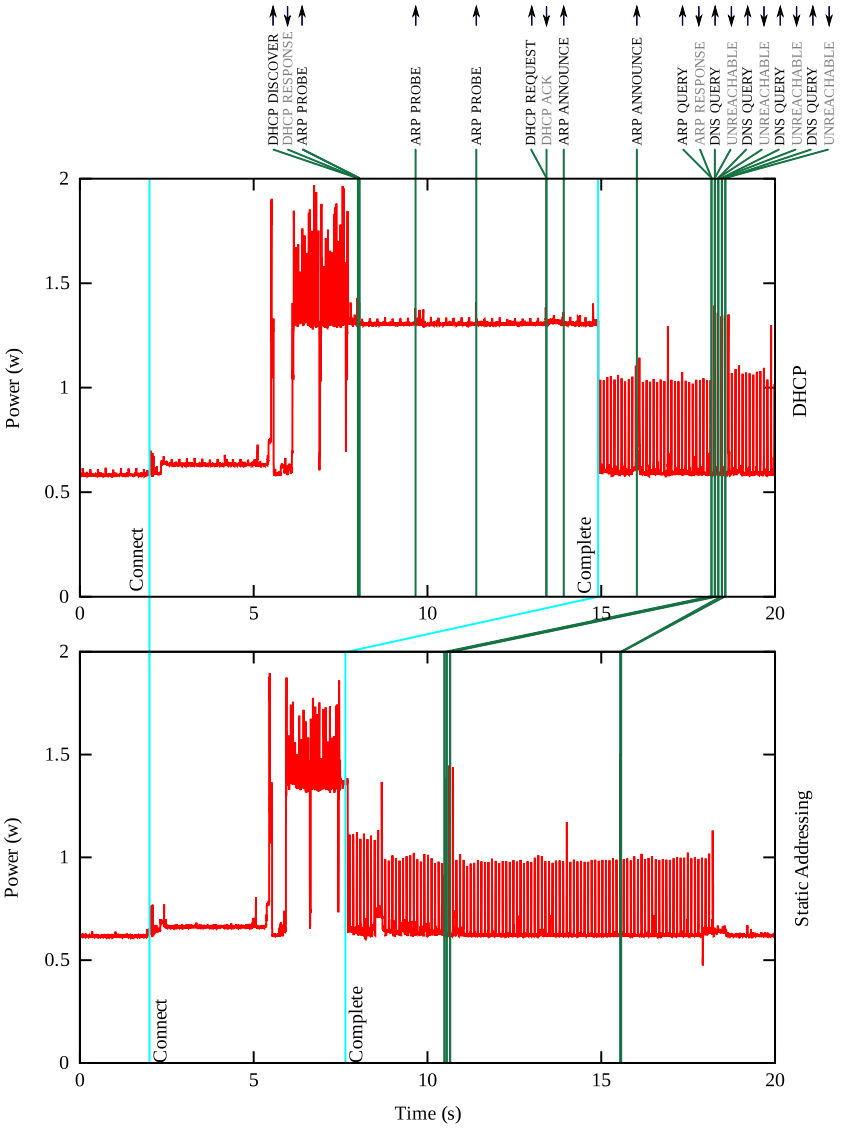

Fig. 9. Energy trace of connecting to Google G1 handset to a WiFi network.

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.

6.1. Connecting to the network

Fig. 9 shows part of the energy trace of connecting to a WiFi network. The top trace shows the cost of obtaining an IP address using DHCP, annotated using the method described with each IP packet sent and received. The bottom trace shows the same operation but using static addressing. 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 ability to identify and account for these occurrences is another advantage of the annotation system. 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).

A considerable energy saving is available from the use of static addressing. The histogram in Fig. 10 shows a summary of the energy consumed in 200 trials of each of these two techniques. The most common cost for a dynamic connection

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

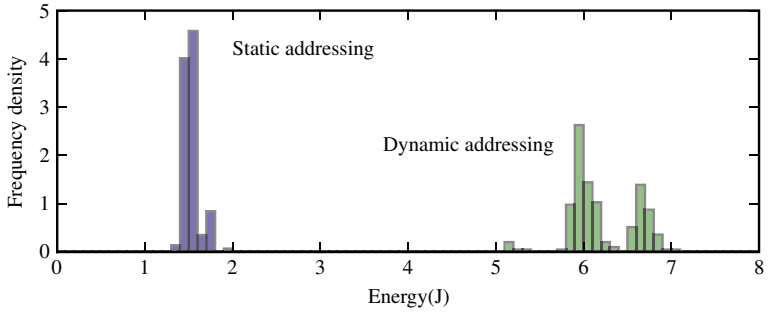

Fig. 10. Energy consumed by a G1 handset connecting to the wireless network.

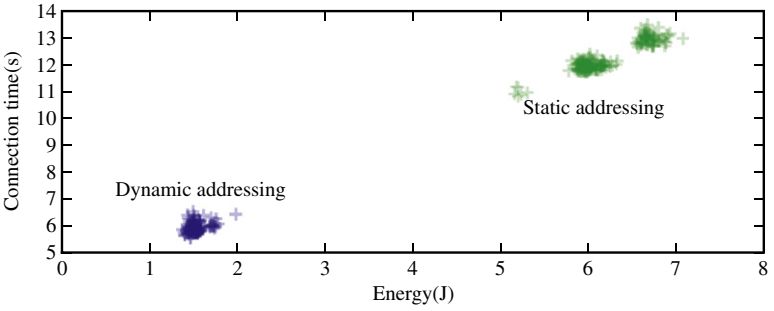

Fig. 11. Energy against time-taken to connect to a wireless network.

is around 6 J whereas a static connection commonly consumes only 1.5 J. It is also notable that the energy consumed by dynamic addressing looks to be partitioned into a number of separate distributions. This is in fact due to discontinuities in the time taken for a connection to complete as shown in Fig. 11. The cause of this phenomena is not yet clear. One possible explanation is that of a timeout in a polling loop in the operating system or the power control hardware in the wireless chip. The clusters we see in Fig. 11 are at approximately 1 s separation—one could imagine this as an appealing number to a developer who needs to select a value for a timeout or sleep period.

One means to improve the energy efficiency of connection whilst maintaining the flexibility of dynamic addressing is to eliminate the ARP probe stage from the process. This is permitted by the RFC in specific situations [7]. In fact this optimisation has now been applied in later versions of the Android operating system. Fig. 12 shows the connection traces for a Google Nexus handset connecting to a wireless network.

This saving is clearly evident in the histogram of connection cost for the Nexus handset (Fig. 13). We note also that the use of static addressing on this handset continues to present an energy saving albeit highly reduced.

6.1.1. Energy saving in context

Owners of G1 handsets might consider switching to static addressing. The typical saving in this case would be around 5 J. This is approximately equivalent to

- 5 s *of talk time*. The average power consumption when in a call seems to be around 1 W;
- 500 s (8 min) *of standby time*. The average power consumption with the phone in standby is around 0.01 W (Fig. 4);
- 200 s (3.5 min) *of idle WiFi connection*. An idle WiFi connection adds around 0.024 W to the phone's consumption (Section 6.2).

Alternatively, a device which makes a connection every 10 min (for example polling for new email) therefore makes around 144 connections a day. If we use a nightly charging strategy with a typical battery of 1400 mAh at 3.7 V then the saving corresponds to around 4% of the total battery life of the handset.

6.2. Idle power

Fig. 14 shows excerpts of the energy trace of a handset when connected to only the 3G, 2G or WiFi networks. The trace shows only the consumption of the wireless networking hardware because the baseline power consumption of the phone (CPU, backlight etc.) have been subtracted from the trace. 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 100 ms, corresponding to receiving base station beacons), the base power is lower than maintaining a connection to the cellular network.

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

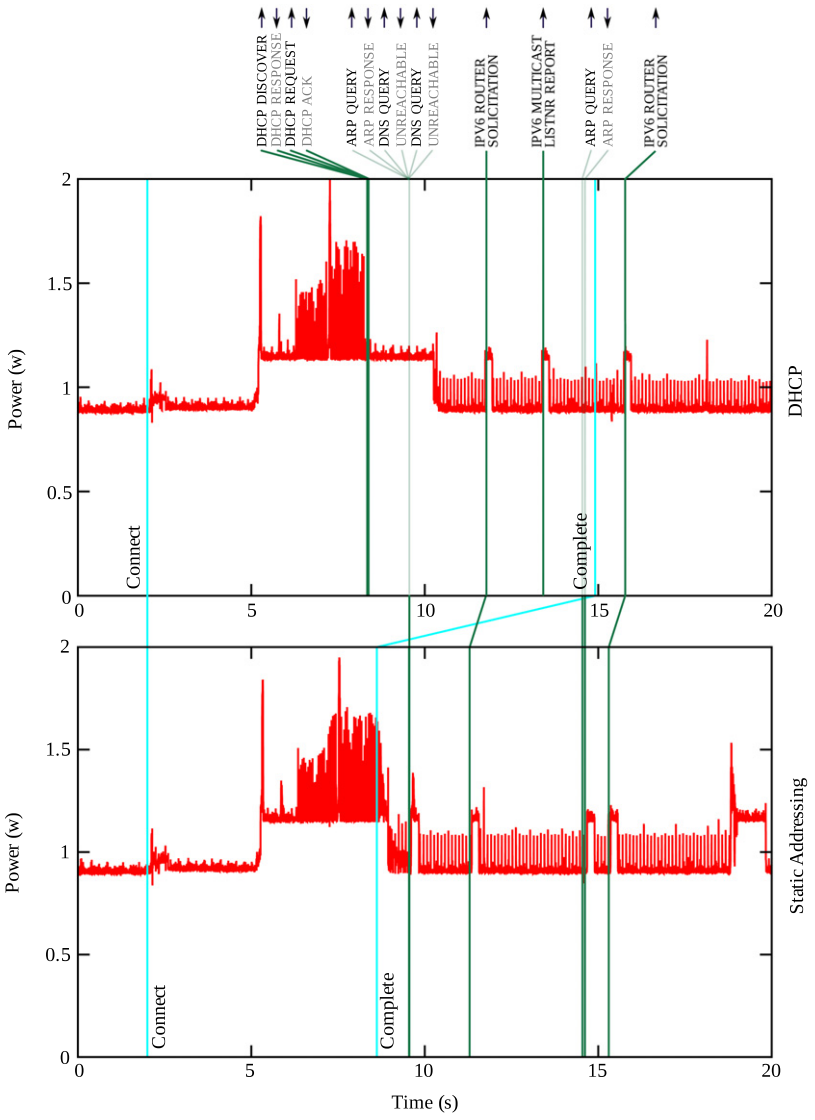

Fig. 12. Energy trace of connecting a Google Nexus handset to a WiFi network.

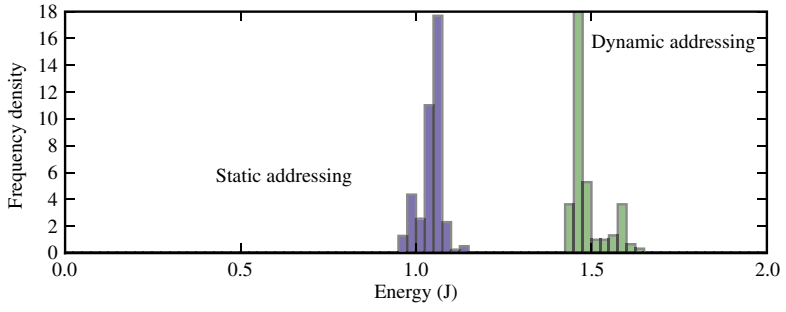

Fig. 13. Energy consumed by a Nexus handset connecting to the wireless network.

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.

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

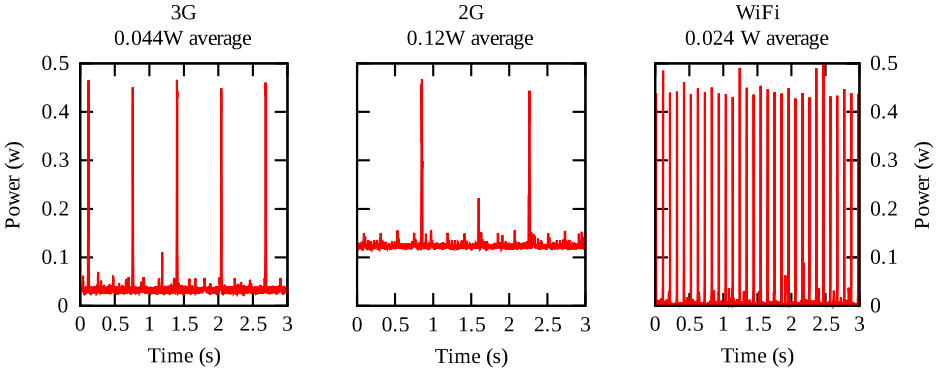

Fig. 14. Additional power consumption incurred when connected to 3G, 2G and WiFi networks.

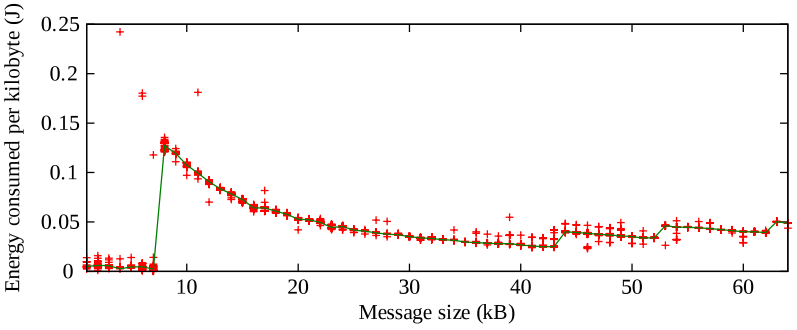

Fig. 15. Variation in G1 energy cost per unit data with total message size.

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 but in this particular case it is the highest power option.

6.3. Data transmission

One might intuitively expect that the energy cost of sending a byte of data reduces, or at worst remains constant, as the total amount of data sent increases. However, this is not the case. Fig. 15 shows the number of Joules required to transmit each kilobyte of data for increasing total message size using the G1 handset. 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 kB 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 look ups of particular Google servers. The evidence of this activity is visible in the packet trace collected by the Power Server. 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. Fig. 16 shows these two instances in more detail. When sending 8 kB or more of data (Fig. 16(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 (Fig. 16(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[7] 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. These results are also almost identical when using the HTC Magic handset and the HTC Hero handsets.

The network proximity of our test server process to the phone means that our results are for a TCP connection with very low round trip time (RTT). This fact combined with the high packet processing speed of our test server means that the phone

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

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

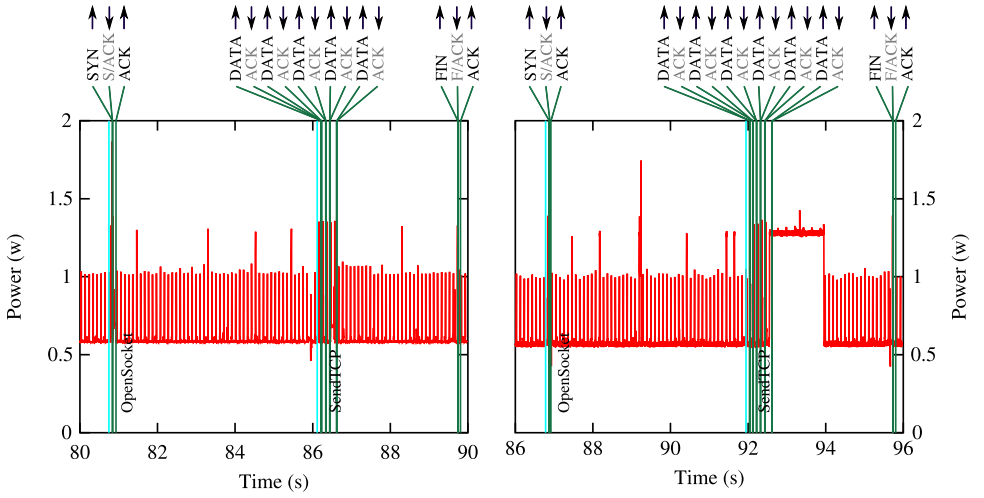

Fig. 16. Extracts from the G1 energy traces of sending 7 kB (left) and 8 kB (right) of data over WiFi.

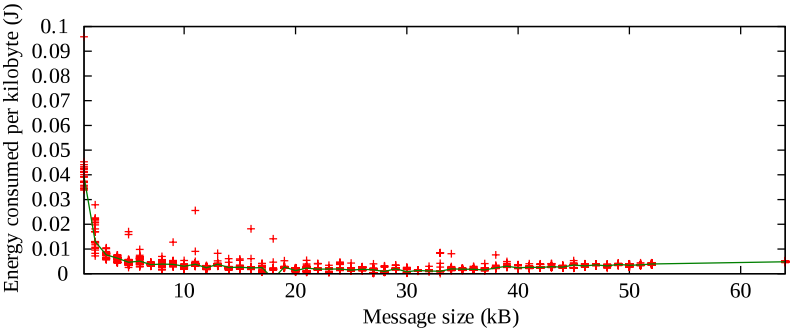

Fig. 17. Variation in N1 energy cost per unit data with total message size.

rarely manages to send a second packet before the acknowledgement of the first packet arrives. We validated this assertion by running a set of automated tests with varying connection latency. High latency connections (of the order of 100 ms RTT) show conventional TCP slow start behaviour.

Fig. 17 shows the variation in energy cost for the Nexus handset. The trend shown in this graph conforms much more to our expectations in which longer messages are more efficient than shorter ones. When compared with the G1 results (Fig. 15), we see that the best case cost for both handsets is around 0.005 J/kB of data, whereas the worst case for the G1 is approximately 0.13 J/kB and the worst case for the N1 is approximately 0.04 J/kB. These extrema occur in different places on the graph for the two handsets. In fact a message size of 8 K is close to the best efficiency on the N1 and the worst on the G1.

6.4. 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 are sent by getting the OutputStream object associated with a Socket instance. Data are then sent over the network by calling the write method on the socket and passing an array of bytes to send.

Nagle's algorithm[8] is used in TCP to aggregate small data chunks into a single larger packet. However, this occurs only when there is unacknowledged data in transit. Given the small RTT of our test setup, this is rarely the case and so the byte array is sent immediately to the client socket without waiting for further data. The size of this array therefore causes significant changes in energy costs. Fig. 18 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 10-fold difference in the energy cost!

[8] Described in RFC896.

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

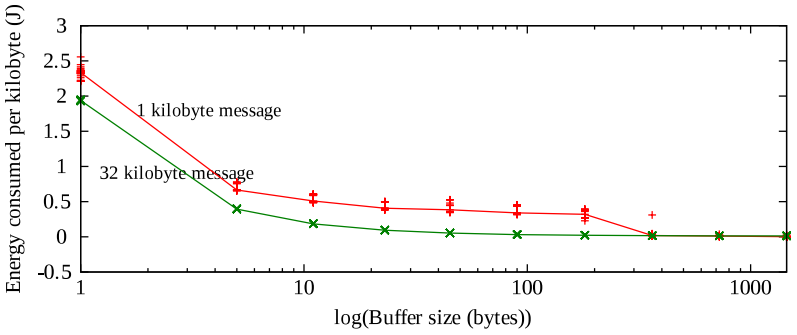

Fig. 18. Variation in energy cost per unit data with buffer size.

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, the 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 7 K/8 K 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 are in them, and with a large enough send buffer there are sufficiently few frames that the 1 kB message does not cross this barrier.

A send buffer of 1448 bytes maximally fills the payload of a TCP packet when sent over Ethernet. We find our best (and most consistent) performance results when using a send buffer of this size. If the buffer is smaller than this, then transmission efficiency is lower because packets are sent only partially filled. If the buffer is larger than this, then the operating system must fragment data over multiple packets—this operation also incurs an overhead and reduces transmit performance.

6.5. 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 show 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. The technique of calculating a baseline power consumption after loading the CPU to stabilise the power consumption of the device means that our framework is unable to measure the power implications of variations in CPU load during a test. It is best suited to measuring the consumption of peripheral devices in the phone such as networking hardware, sensors, camera and audio devices.

The implications of the results we have presented in this work are as follows.

- Owners of G1, Magic and Hero handsets would be well served to switch to static addressing if they make frequent use of wireless networks which they administer.
- Owners of Nexus handsets will see only a minimal benefit from the use of static addressing.
- Developers on Nexus handsets may assume that efficiency increases with message size but developers on G1, Magic and Hero handsets should limit data transmission over the WiFi to 7 K at a time if possible.
- It should not be assumed that the relative power consumption of different networking technologies remains constant over time. In fact this relative consumption changes between hardware generations too: in 2007, it was reasonable to assume that the WiFi network was an order of magnitude more expensive in idle mode than the cellular network [8]. For modern handsets, this is no longer the case.
- Developers should be aware of the implications of layering in the architecture of networks. Selecting a send buffer size which matches the maximum transfer unit of the network can produce a significant energy improvement over small buffers which under-fill packets and over large buffers which incur fragmentation and reconstruction overheads in the operating system.

7. Related work

Dutta et al. present an ingeniously simple design for energy metering *in situ* by augmenting switching regulators [9], and Fonseca et al. build on top of this hardware platform to apportion energy costs of components in embedded network devices to individual activities [10]. 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 [11] and the PowerScope tool for profiling energy usage [12]. The measurements from these tools summarise the behaviour of a general application whereas our measurements concern the behaviour of an action taken within an application.

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

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 [13]; similarly, Mohan et al. looked at the overall power required by the sensors for their pervasive application but did not investigate any further [14].

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,[9] 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.[10]

8. 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 connecting to an 802.11 wireless network and sending messages over this network. This is a particularly common feature of pervasive computing applications. Our results have shown interesting phenomena but the search space for potential savings remains huge. We are currently constructing a mobile version of our measurement platform in order 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, our results show that the optimisation criteria for minimising power consumption change between devices, operating systems and operating scenarios. 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.

Acknowledgements

We would like to thank Brian Jones, Alastair Beresford and Rob Harle for their advice and guidance, Andy Hopper for his insight and support, and Kieran Mansley for his help on the details of TCP. We also gratefully acknowledge the support of Google Inc. through a Google Focused Research Award and for their donation of G1 and Nexus handsets.

References

[1] International Telecommunications Union, World telecommunication/ICT indicators database, Technical Report, 2008.
[2] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, M.B. Srivastava, Participatory sensing, in: WSW'06 at SenSys'06, 2006.
[3] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, P. Boda, PEIR, the personal environmental impact report, as a platform for participatory sensing systems research, in: Mobisys'09: Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services, 2009.
[4] J.J. Davies, A. Beresford, A. Hopper, Scalable, distributed, real-time map generation, IEEE Pervasive Computing 5 (2006) 47–54.
[5] K. Mansley, A.R. Beresford, D. Scott, The carrot approach: encouraging use of location systems, in: UbiComp'04: Proceedings of the Sixth International Conference on Ubiquitous Computing, 2009.
[6] R. Gold, Optimal binary sequences for spread spectrum multiplexing (corresp.), IEEE Transactions on Information Theory 13 (1967) 619–621.
[7] D. Plummer, Ethernet address resolution protocol, RFC 826, Standard, 1982.
[8] A. Rahmati, L. Zhong, Context-for-wireless: context-sensitive energy-efficient wireless data transfer, in: MobiSys'07: Proceedings of the 5th International Conference on Mobile Systems, Applications and Services, ACM, New York, NY, USA, 2007, pp. 165–178.
[9] P. Dutta, M. Feldmeier, J. Paradiso, D. Culler, Energy metering for free: augmenting switching regulators for real-time monitoring, in: ISPN'08: Proceedings of the Seventh International Conference on Information Processing in Sensor Networks, 2009.
[10] R. Fonseca, P. Dutta, P. Levis, I. Stoica, Quanto: tracking energy in networked embedded systems, in: OSDI'08: Proceedings of the 8th USENIX Symposium on Operating System Designs and Implementations, 2009.
[11] K. Farkas, J. Flinn, G. Back, D. Grunwald, J. Anderson, Quantifying the energy consumption of a pocket computer and a java virtual machine, in: SIGMETRICS'00: Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 2000.
[12] J. Flinn, M. Satyanarayanan, Powerscope: a tool for profiling the energy usage of mobile applications, in: Mobile Computing Systems and Applications, 1999, Proceedings, WMCSA'99, Second IEEE Workshop on, 1999, pp. 2–10.
[13] T. Pering, Y. Agarwal, R. Gupta, R. Want, CoolSpots: reducing the power consumption of wireless mobile devices with multiple radio interfaces, in: MobiSys'06: Proceedings of the 4th International Conference on Mobile Systems, Applications and Services, 2006.
[14] P. Mohan, V. Padmanabhan, R. Ramjee, Nericell: rich monitoring of road and traffic conditions using mobile smartphones, in: SenSys'08: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems, 2008.

[9] http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/Plug-ins/Enablers/Nokia_Energy_Profiler/.
[10] http://code.google.com/events/io/sessions/CodingLifeBatteryLife.html.

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005

Andrew Rice is an Assistant Director of Research in the Computer Laboratory at the University of Cambridge. His current research agenda is Computing for the Future of the Planet, investigating the contributions that computer science can make in solving issues such as over-population and climate change. Examples include modelling and simulation, sensing the planet and the future use of mobile phones as a comuting platform.

Simon Hay is a Ph.D. candidate in the Computer Laboratory at the University of Cambridge. He is conducting research into a Personal Energy Meter that collects information about our daily consumption and provides breakdowns of the energy costs of activities to help us target areas for reduction.

Please cite this article in press as: A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.07.005