An Investigation into Energy-Saving Programming Practices for Android Smartphone App Development

Ding Li and William G. J. Halfond
University of Southern California
Los Angeles, California, USA
{dingli, halfond}@usc.edu

ABSTRACT

Developing energy efficient mobile applications is an important goal for software developers as energy usage can directly affect the usability of a mobile device. Unfortunately, developers lack guidance as to how to improve the energy efficiency of their implementation and which practices are most useful. In this paper we conducted a small-scale empirical evaluation of commonly suggested energy-saving and performance-enhancing coding practices. In the evaluation we evaluated the degree to which these practices were able to save energy as compared to their unoptimized code counterparts. Our results provide useful guidance for mobile app developers. In particular, we found that bundling network packets up to a certain size and using certain coding practices for reading array length information, accessing class fields, and performing invocations all led to reduced energy consumption. However, other practices, such as limiting memory usage had a very minimal impact on energy usage. These results serve to inform the developer community about specific coding practices that can help lower the overall energy consumption and improve the usability of their applications.

Categories and Subject Descriptors

D.2.8 [**Metrics**]: Performance measures

General Terms

Performance

Keywords

Energy, Programming patterns

1. INTRODUCTION

Mobile devices, such as smartphones and tablets, have become almost ubiquitous in our daily lives. These devices accompany us constantly and the apps they run provide helpful information and services by combining cloud data and sensor measurements in new and innovative ways. Unfortunately, these devices are limited in terms of their battery power and the extensive usage of sensors and network data can rapidly drain the devices' batteries and limit the usefulness of the device and its apps. Although advances in hardware and battery technology have helped decrease a device's energy consumption, these improvements cannot prevent an inefficient or poorly designed app from needlessly draining the device's battery.

Developing an app so that it is energy efficient is challenging and implementations can vary widely in terms of their energy consumption. As a result, battery usage has become an important, albeit informal, quality metric for marketplace apps. A cursory examination of marketplace reviews shows that many users complain about battery usage and this can inform their decision to give positive or negative ratings to an app. This makes improving energy consumption an important goal for mobile app developers.

There are existing tools that can help developers to gain insight into the energy usage patterns of their applications. Examples of such techniques are cycle-accurate simulators [1, 2], power monitors [3], program analyses [4, 5], and statistical based measurement techniques [6]. Although these techniques allow developers to understand where energy is consumed within their application (e.g., by which source lines), they do not provide direct guidance as to how to improve the app's energy consumption. That is, they do not address the gap between understanding where energy is consumed and understanding how the code can be changed to reduce the energy consumed.

The connection between observed energy consumption and opportunities for energy optimization is not always straightforward. For example, although a particular method may consume a lot of energy, there may not be any alternative implementation mechanisms for that functionality that consume less energy. At the same time, there may be another location that consumes less energy, but has alternative implementation mechanisms that consume even less energy. This situation can make it difficult for developers to readily identify areas for code improvement. To find energy saving best practices, developers can make use of conventional wisdom, consult development blogs written by fellow software engineers, or simply search online for tips. In our own online searches, we found many such sites offering development advice. Unfortunately, many of these suggestions are not supported by empirical evidence and it is not clear how effective they will be in practice.

GREENS'14, June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2844-9/14/06...$15.00
http://dx.doi.org/10.1145/2593743.2593750

In this paper we present the results of a small-scale investigation into the effectiveness of commonly recommended energy saving best practices. For this investigation, we have identified a collection of developer-oriented tips and recommendations for reducing the energy consumption of mobile app implementations. We evaluated these suggestions in a controlled environment to determine if they represented effective and useful guidance for developers. The topics we investigated included optimizing the sending of HTTP packets, memory usage, array lengths, static invocations, and field access.

The results of this evaluation were informative and provide practical guidance for developers. Our experiments showed that sending larger files is more energy efficient than sending smaller files. Case in point, the energy consumption of downloading 1,000 bytes of data is roughly the same as downloading 1 byte. Based on this insight, developers could bundle several small HTTP requests into larger ones to achieve more energy efficiency. We also found that high memory usage will increase the energy consumption of applications but only by a small amount. In our experiment, the average energy consumption per computational step increased 21% for each memory increase of 10,000%. This result indicates that developers should use more energy for data structures, such as network buffers, if it could increase performance or reduce the energy consumption of another component. Finally, we verified that commonly suggested performance oriented best practices regarding array lengths, static invocations, and field access also work for reducing energy. In our experiment, avoiding references to array length in a loop reduced energy by 10%, static invocations consumed 15% less energy than virtual invocations, and direct field accesses used 30-35% less energy than indirect getter and setter methods.

The remainder of this paper is organized as follows. In Section 2 we give an overview of the energy consumption conventional wisdom that we examine in the evaluation. Section 3 provides the details of the experiments, describing the specific hypotheses we considered, experiment setup, and results. We discuss related work in Section 4. Finally we summarize our conclusions and discuss future directions in Section 5.

2. OVERVIEW

In this paper, we consider the energy savings that can be achieved by using different coding practices that are commonly suggested or proposed in the official Android developers web site [7]. Among the many recommended practices, we focused on three broad categories: network usage, memory consumption, and low-level programming practices. We were interested in the network and memory energy because these categories represent two of the most heavily used and energy consuming components in smartphones. We were interested in a range of programming practices that were suggested for performance improvement because these practices are easy for developers to utilize in their code and do not require high-level design changes. Within each category we examined one or more recommended best practices.

We evaluated the best practices for energy savings via controlled experiments. For each of the three categories that we were interested in, we created experiments with and without different programming practices and measured their energy consumption. We then compared the measured results to evaluate the effectiveness of the best practices for saving energy.

The first category of energy saving best practices deals with the energy of networks. Accessing the Internet is one of the most important capabilities of a smartphone and network communication is well known as one of the most energy intensive components in Android smartphones. Particularly, we are most interested in the practices of making HTTP requests. For modern Android smartphones, HTTP requests are one of the most important among many methods for accessing the Internet. Case in point, one of the most common use cases of modern Android smartphones is to visit various web applications. These web applications have to be accessed through HTTP requests. Furthermore, many modern Android application adhere to the REST architectural style and they treat remote resources as URL links. Thus, these RESTful applications need to make HTTP requests to visit remote resources and databases. Unfortunately, although HTTP requests are common and consume a high amount of energy, there are no practical programming guidelines for developers to make energy efficient HTTP requests or design appropriate data transmission protocols based on HTTP requests.

Previous work [8] shows that transmitting large data chunks on the Internet is more efficient than transmitting small data chunks in terms of network throughput. However, there is no evidence to indicate whether the same property holds for energy. In our empirical study in Section 3.2, we show that a larger data size is also more energy efficient than small pieces of data for downloading files via HTTP GET requests. Our experiment indicates that developers should bundle small HTTP GET requests to save energy.

Our second category of interest deals with practices to save memory energy. Specifically, we study whether high memory usage will cause the application to consume more energy. As described by the official Android developers blog [7], memory is not free and developers should avoid unnecessarily allocating memory. However, in many cases, memory allocation is more about trade-offs rather than the yes or no answer about whether it is necessary. For example, it is difficult for developers to choose the proper cache size for their data. High memory usage consumes more allocation energy and may cause the application to become more energy consuming. On the flip side, insufficient memory may cause more energy consumption in other components of the smart phone, such as 3G or WiFi network. Thus, developers need to have guidelines to help them decide what is the appropriate amount of memory to allocate.

As our result in Section 3.3 show, high memory usage contributes trivially to the increase in application energy consumption. Therefore, developers would be better off increasing memory usage energy to save energy consumption by other components.

The third category considers whether the programming tips to improve runtime performance also work for reducing energy consumption. The official Android developers blog [7] has provided some useful tips to improve the performance of Android applications. However, there is no evidence to show that these tips work for energy consumption. Furthermore, as previous work [9] shows, the improvement of performance does not necessarily cause the improvement of energy consumption. Thus, we need to verify whether these tips work for saving energy consumption as well.

Among all the performance tips, we select three of them that are straightforward and easily applied in practice. The first tip is to avoid using *.length* while running through an array. For example, developers should use Program 3 rather than Program 4. The second tip is to access fields directly instead of using an encapsulating method. The third tip is to use static methods, when possible, instead of virtual methods. In our study, we find that these tips for performance also work for saving energy. Together these tips can reduce energy consumption by 10% to 30%.

3. EVALUATION

In this section we describe the experiments we undertook to evaluate the different guidelines and suggestions discussed in Section 2. Broadly, we categorize the experiments as addressing the three following research questions:

RQ1: Will bundling of small HTTP requests save energy?

RQ2: Does high memory usage make the application consume more energy?

RQ3: Can the use of performance-oriented programming tips also serve to reduce energy consumption?

3.1 Experiment Protocol

Our experiments were performed on a Samsung Galaxy II running Android 4.2.2. For network access, the phone was connected to a local WiFi network. The energy consumption on the phone was measured by the Monsoon power meter [10]. The power meter was connected and controlled by a Dell XPS 8100 desktop running Windows 7 Professional with a 3GHz Intel i5 processor and 8GB memory. The HTTP server process was running on the same desktop. To account for random variations in the underlying mobile device and power meter, we performed each data collection five times and averaged the collected data. Energy values are expressed in milliampere-hour (mAh).

3.2 RQ1: HTTP Request

To address RQ1 we measured the energy consumption of downloading different sized data files via HTTP requests.

```
 1 protected void onCreate(Bundle savedInstanceState) {
 2    super.onCreate(savedInstanceState);
 3    setContentView(R.layout.activity_main);
 4    URL url = new URL(resourcelink);
 5    for(int i=0;i<N;i++)
 6    {
 7       URLConnection urlConn = url.openConnection();
 8       InputStream in = new BufferedInputStream(urlConn.
              getInputStream());
 9       try {
10           readStream(in);
11       finally {
12           in.close();
13       }
14    }
15    finish();
16 }
```

Program 1: Test harness for network measurements

3.2.1 *Experiment Design*

To carry out the experiment, we first created the small program shown in Program 1 to access the different sized data files from the server. Lines 1–3 are infrastructure code to set up the task so it would run on the mobile platform. At line 4, we prepare the URL links that point to the data files on the server. At lines 5-14, we run the network request N times (we set $N = 1,000$). At line 7, we send the request for the target file. At line 10, we read all the received data without any processing. We read all of the data because we want to ensure that the system has downloaded all of the transmitted file without leaving any in a buffer, since our experiments rely on accurate data file size measurements. At line 15, we finish the task and destroy the current activity.

For the network access function at line 7, there are two types of APIs in Android to make an HTTP request. The first one is to use `HttpURLConnection` class and the second one is to use `HttpClient` class. Both of these classes are widely used in market applications and support all of the methods defined in the HTTP protocol, such as GET and POST. However, recent posts in the Android Official Developer's Blog [11], suggest that `HttpURLConnection` is preferred over `HttpClient`. Thus, in our preliminary study, we only measure the energy consumption of `HttpURLConnection`. In particular, we measure the energy consumed by the HTTP GET request method.

In our experiment, we first ran the program on the mobile device with an empty body loop for lines 5–14. We used this run to calculate a baseline energy for the loop and to better isolate the cost of the network access. Then we ran the program and downloaded a set of files from the server that ranged in size from one byte to 4,000 bytes. For each download, we measured the energy and subtracted out the baseline energy to calculate the cost of the network access.

3.2.2 *Result Report and Summary*

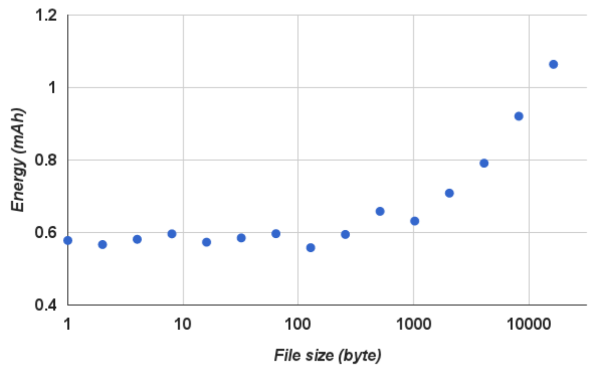

Figure 1: Energy cost of downloading the different sized data files.

The results of our experiment are shown in Figure 1. The y-axis of Figure 1 is the energy consumption of the HTTP GET request in units of *mAh*. The x-axis is the number of bytes that are downloaded by the request. From Figure 1, we can see that the energy consumption of an HTTP request stays in the range of 0.5-0.6 *mAh* when the downloaded data is less than 1,024 bytes (1,000 on the x-axis in Figure 1). When the data size becomes larger, the energy consumption of the HTTP GET request has a roughly linear relationship with the size of downloaded data.

The reason for this observed behavior is that the HTTP protocol and its lower level network protocols, TCP and IP, have a fixed overhead introduced by the packet header and control information. The size of the packet header is fixed without regard to the size of the transmitted data. The size of the control information mainly depends on the number of packets rather than the packet size. Thus, energy consumption is dominated by transmitting packet headers and control information when the packet size is small. As the amount of data increases, the situation reverses and the data size dominates. This inefficiency of sending small packets is also observed when investigating network throughput [8].

The result of this experiment indicates that bundling small HTTP requests could save energy and that developers should avoid sending small HTTP requests. Especially for some RESTful applications, developers should avoid querying remote databases for just a few bytes. If it is necessary to make multiple small HTTP requests, developers should try to design their apps or protocols so that these can be bundled into one larger request to save energy.

In this experiment we have focused on HTTP GET downloads since our observations of many mobile apps indicate that individual download requests occur with a high frequency. Nonetheless, both uploads and downloads represent opportunities for energy improvements and we plan to investigate uploads in future work, as well as investigate if there are differences between GET and POST requests.

3.3 RQ2: Use of Memory

We address RQ2 by measuring the average energy consumption of accessing different heap-based objects that represent different levels of memory usage.

3.3.1 Experiment Design

```
 1 protected void onCreate(Bundle savedInstanceState) {
 2     super.onCreate(savedInstanceState);
 3     setContentView(R.layout.activity_main);
 4     int[] array=new int[size]
 5     for(int k=0;k<N;k++)
 6         for(int i=0;i<size;i++)
 7         {
 8             array[i]=1;
 9          }
10 }
```

Program 2: Test harness for memory measurements

Program 2 shows the test harness for our experiment. The size of the array at line 4 reflects the amount of memory that is allocated to the application. For our experiments, we created an array of integers with size from 512 to 5,120,000 (line 4). Then, we accessed the array (line 6 to line 9). To ensure that the energy consumption of the loop dominates the total energy, we also ran the loop N times (line 5). In our experiment we set $N = 100,000$.

In the results, we report the average energy consumption per each access to the array units. That is $AE = \frac{E}{N * size}$, where AE is our reported result, E is the measured energy consumption of the whole program, $size$ is the size of the memory allocated, and N is the loop iteration count.

3.3.2 Result Report and Summary

Our measured results are shown in Figure 2. The y-axis is the energy consumption per each access of each array cell in units of $10^{-5} mAh$. The x-axis is the array length.

From Figure 2, we can see that the average energy consumption per each access to an array cell increases when the memory usage gets higher. However, this increment is very

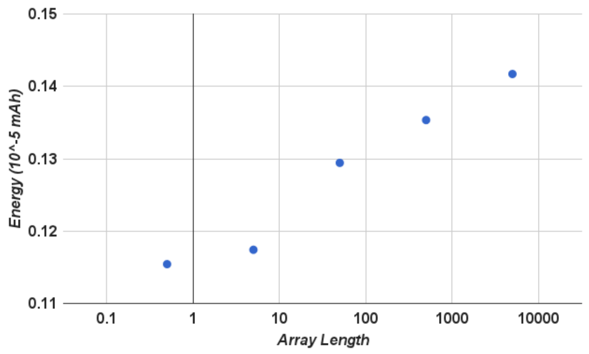

Figure 2: Energy consumption at different levels of memory usage

small. Notice that our y-axis starts from $0.11 * 10^{-5} mAh$ and our x-axis is in log scale. The results show that the energy consumption per each access to an array increases 21.7% while, at the same time, the memory usage increases 1,000,000%.

The experiment's results indicate that even though memory is not free and developers should avoid allocating unnecessary memory, they are not as expensive as thought. Higher memory usage only slightly increases the average energy consumption of each access. This result can influence development practice by encouraging developers to allocate more memory if it could help save energy for other components. For example, developers could allocate a larger cache to reduce the number of accesses to the network.

3.4 RQ3: Performance Tips

```
Object[] array;
int l=array.length;
for(int i=0;i<l;i++)
{
    array[i]=null;
}
```

Program 3: The recommended loop structure

```
Object[] array;
for(int i=0;i<array.length;i++)
{
    array[i]=null;
}
```

Program 4: The baseline loop structure

To address RQ3, we compared the energy consumption of code that was and was not implemented using best practices oriented towards runtime performance. In particular we looked at the three suggestions explained in Section 2. These were (1) avoiding access of an array's length attribute in the body of a loop, (2) accessing fields directly instead of through getters and setters, and (3) using static instead of virtual invocations. For each of these practices, we implemented code that either followed or did not follow the recommendation and then compared the energy consumption of those programs.

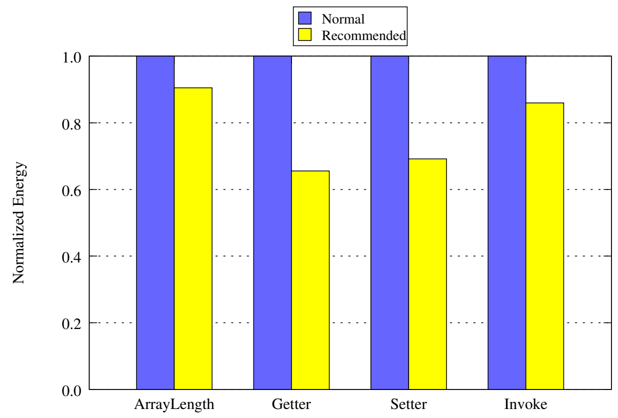

Figure 3: Relative energy consumption of the performance-oriented best practices.

```
1 protected void onCreate(Bundle savedInstanceState) {
2    super.onCreate(savedInstanceState);
3    setContentView(R.layout.activity_main);
4    for(int i=0;i<N;i++)
5    {
6      Action();
7    }
8    finish();
9 }
```

Program 5: Test harness for evaluating the performance oriented best practices.

3.4.1 Experiment Design

Our test harness for the experiments is shown in Program 5. The *Action*() at line 6 is a placeholder for the different actions that we need to perform for the different experiments, which are detailed below. We ran each action for $N = 50,000,000$ times to get a stable measurement. We also measured the energy consumption with no instructions in *Action* and subtracted this energy as a baseline energy to arrive at the final result.

Array Length: To measure the energy consumption of the array length operation in the loop, we performed the following operation. First, we inserted the code in Program 3 at the position of *Action*() in Program 5 and measured the test harness' energy consumption. Then we repeated this process but used the code in Program 4 instead.

```
class obj{
  public int a=11;
  public void Setter(int b)
  {
    this.a=b;
  }
  public int Getter()
  {
    return this.a;
  }
}
```

Program 6: The test code for Getter/Setter

Field Access: To measure the energy consumption of the direct field access versus using the getters and setters, we performed the following operation that resulted in four different programs. First we created the object shown in Program 6. Then we created two test programs based on replacing *Action*() in Program 5. The first of these reads and writes the field *obj.a* directly and the second accesses the field *obj.a* via the methods Getter and Setter. For each program we measured the energy consumption of the test harness.

```
 1 class methods{
 2   public int Foo()
 3   {
 4     return 0;
 5   }
 6   public static int StaticFoo()
 7   {
 8     return 0;
 9   }
10 }
```

Program 7: The test code for invocation

Invocation: To measure the energy consumption of different invocation mechanisms, we set the *Action*() of Program 5 to call a method that directly returns an integer value. Then we created two variants, one used a virtual method, such as *Foo*() at line 2 of Program 7, and the other used a static method, such as *StaticFoo*() at line 6 of Program 7. We measured the energy consumption of these two variants.

3.4.2 Result Report and Summary

The results of our experiment are reported in Figure 3. All results are normalized to the value of the *Normal* column in each column pair. The columns labeled *ArrayLength* are our experiment results for the array length operation in loops. *Normal* means the energy consumption of the loop in Program 3 and *Recommended* means the energy consumption of the loop in Program 4.

The columns *Getter* and *Setter* are the results for accessing the fields of objects. Column *Getter* is the result of reading the value of the field and *Setter* is the result of setting the field to a new value. *Normal* means we access the field by a method and *Recommended* means we access the field directly.

The column *Invoke* is the result for the recommendation to use static invocations instead of virtual invocations. *Normal* is the energy consumption of a virtual invocation and *Recommended* is the energy consumption of a static invocation.

Array Length: In Figure 3, the energy consumption of loops that do not refer to array length is 10% less than in the cases when the loop refers to array length. As mentioned in the Android developer blog [7], a reference to the array length is not free in Android. Our results show that avoiding references to the array length for each loop iteration can save energy. For developers this suggests that initializing a variable with the length once and then using that as a constant could reduce the energy consumption of loops.

Field Access: As shown in Figure 3, direct access to object fields consumes 35% and 31% less energy, respectively, for reading and setting field values. According to the Android developer blog [7], the reason is that the virtual methods that are used to access field values are expensive operations in Android. Thus, developers should directly access the fields of objects rather than through a method if they are trying to save energy. However, this should be balanced with the need to maintain good encapsulation in the design.

Invocation: As shown in Figure 3, the energy consumption of static invocations is 15% less than virtual invocations. This result indicates that static invocation is more efficient in Android. This is likely due to the lookup overhead incurred with virtual method invocations. Although always using static methods is not a good programming practice, it is something developers should consider when trying to save energy in code sections that repeatedly invoke other methods.

4. RELATED WORK

Previous efforts to provide energy information or guidelines to developers mainly provide a fine-grained energy estimation or measurement. Cycle-accurate simulators [1, 2] simulate the low-level behavior of the CPU. Therefore they can estimate the energy consumption of the system in a very detailed manner. The drawback of cycle-accurate simulators is that they are extremely slow. Other works model the software energy at the instruction level [12, 13, 14, 15]. These works build a model for assembly instructions or micro-instructions and estimate the energy consumption of software applications based on this instruction model. They generally only provide estimates for CPU energy consumption and do not take the operating system into consideration. System state energy modeling [16, 17] estimates the energy consumption of applications by monitoring the state transitions of different hardware components in an embedded system. System routine level energy modeling [18, 19] builds energy functions of operating system routines to estimate the energy consumption of systems. Our previous work [4, 5, 6] estimates and calculates the energy consumption of Android applications by combining path profiling techniques with energy models and power meters.

All of the works mentioned above focus on providing accurate numeric estimates of the energy consumption at different program levels, such as code blocks, methods, or source lines. However, they do not provide general high level guidance to developers in terms of energy efficient practices in programming.

Another group of work maps the energy consumption to certain components in software architecture. Sahin and colleagues [20] build a framework to map energy consumption to software design patterns. Their tool can monitor the energy consumption of each component in the software system. Furthermore, they also find that the energy consumption of a software system is related to its number of objects and the amount of messages passed among objects. Litke and colleagues [21] measure the energy consumption savings of using three software design patterns, factory creational pattern, structural pattern, and behavioral pattern. Their result shows that applying certain software patterns could save energy for software systems. Seo and colleagues [22, 23] estimate the energy consumption of software architecture components by monitoring Java bytecodes in a JVM. Although these approaches provide high level information about energy consumption on the software architecture level, they do not provide concrete guidelines to help developers implement more energy efficient code.

The closest work to our paper is Vetro' and colleagues' work [24]. They defined energy code smells as an implementation choice that makes the software less energy efficient. They also validated several energy code smells, such as self assignment, dead local store, and useless control flow. These smells are generally redundant code or operations that can also be detected or eliminated by current compiling techniques. Unlike this work, the patterns studied in our paper are more specific to energy consumption. Our patterns do not focus on any redundancy in source code but still cause the software to be less energy efficient.

Bunse and colleagues [9] compared the energy consumption of different sorting algorithms. Their results show that even though slower sorting algorithm, such as insert sort, may take longer time to run, they consume less power than faster sorting algorithms, such as quick sort. Thus, developer should choose appropriate sorting algorithms for their energy context. However, this work only compares the energy consumption of sorting algorithms. It does not provide a general guideline for other types of applications or algorithms.

Manotas and colleagues [25] compare the energy consumption of web applications on different web servers. They measure the energy consumption of seven web applications on five different web servers. Their experiment shows that the energy consumption of web applications is highly dependent on the web servers that they run on. Their work focuses on the energy consumption of web servers rather than applications and does not provide any programming guidelines for how to create energy efficient web applications.

Ardito and colleagues [26] compared the energy consumption of two Android phones of different generations, Samsung Galaxy I7500 and Samsung Nexus S. They concluded that the Samsung Nexus S is more energy efficient than its previous model, the Samsung Galaxy I7500.

Besides work that provides information or guideline about energy consumption to developers, there are others that focus on optimizing energy consumption of mobile devices. Chameleon [27] saves energy for apps running on devices with OLED screens by manually changing bright colors to dark colors. Their approach reduces the energy consumption of the OLED screens by 60%. Our recent work [28] used automated program analysis techniques to automatically improve mobile web apps and take advantage of this insight. Su and colleagues [29] reduce the energy consumption introduced by state switches of the CPU by reordering the compiled instructions. Davidson and colleagues [30] reduce the energy consumption of memory by minimizing memory access. Mehta and colleagues [31] reduce the energy consumption of processors by carefully labeling the instruction registers to reduce state switches inside the processor. All of these works focus on reducing the energy consumption of a certain part of the system rather than providing general guidelines for developers' coding practices.

5. CONCLUSION AND FUTURE WORK

Developing energy efficient mobile applications is an important goal for software developers as energy usage can directly affect the usability of a mobile device. Unfortunately, existing energy-oriented techniques tend to focus on understanding where energy is consumed within an application and how much is consumed. Additionally, most online resources tend to be oriented towards improving runtime performance and provide suggestions that do not necessarily lead to reductions in energy usage. The resulting situation is that developers lack guidance as to how to improve the energy efficiency of their implementation and which practices are most useful.

In this paper we conducted a small-scale empirical evaluation of commonly suggested energy-saving and performance-enhancing coding practices. In this evaluation we evaluated the degree to which these practices were able to save energy as compared to their unoptimized code counterparts. Our investigation focused on several different types of practices: network packet size, memory usage, array length access, direct field access, and invocation type. Our results were informative. In general, we found that many of these practices could also reduce energy consumption. For example, bundling network packets up to a certain size and using certain coding practices for reading array length information, accessing class fields, and performing invocations all led to reduced energy consumption. However, other practices, such as limiting memory usage had a very minimal impact on energy usage. These results serve to inform the developer community about specific coding practices that can help lower the overall energy consumption of their mobile apps and improve the usability of their systems.

In future work we plan to significantly expand the scope of the paper's study. In particular, we plan to investigate whether the patterns identified here are consistent across multiple mobile platforms and operating systems and include additional best practices in the evaluation that have been suggested by the software engineering community. We also plan to increase the depth of each investigation by increasing the number of data points to improve statistical validity measures and examining the effect of more independent variables within each research question.

6. ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under Grant No. CCF-1321141.

7. REFERENCES

[1] T. Mudge, T. Austin, and D. Grunwald, "The Reference Manual for the Sim-Panalyzer Version 2.0."

[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-level Power Analysis and Optimizations," *SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 83–94, May 2000.

[3] D. McIntire, K. Ho, B. Yip, A. Singh, W. Wu, and W. J. Kaiser, "The Low Power Energy Aware Processing (LEAP)Embedded Networked Sensor System," in *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, ser. IPSN '06. New York, NY, USA: ACM, 2006, pp. 449–457.

[4] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating Android applications' CPU energy usage via bytecode profiling," in *Proceedings of the 1st International Workshop on Green and Sustainable Software (GREENS)*, May 2012, pp. 1–7.

[5] ——, "Estimating Mobile Application Energy Consumption Using Program Analysis," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 92–101.

[6] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan, "Calculating Source Line Level Energy Information for Android Applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ser. ISSTA 2013. New York, NY, USA: ACM, 2013, pp. 78–89.

[7] "http://developer.android.com/training/articles/perf-tips.html."

[8] I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside Dropbox: Understanding Personal Cloud Storage Services," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, ser. IMC '12. New York, NY, USA: ACM, 2012, pp. 481–494.

[9] C. Bunse, H. Höpfner, S. Roychoudhury, and E. Mansour, "Choosing the" Best" Sorting Algorithm for Optimal Energy Consumption," in *Proceedings of the 4th International Joint conference on Software Technologies*, 2009, pp. 199–206.

[10] Masoon, "http://www.msoon.com/labequipment/powermonitor/."

[11] "http://android-developers.blogspot.com/2011/09/androids-http-clients.html."

[12] A. Sinha and A. P. Chandrakasan, "JouleTrack: A Web Based Tool for Software Energy Profiling," in *Proceedings of the 38th Annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 220–225.

[13] S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel, "An Accurate and Fine Grain Instruction-level Energy Model Supporting Software Optimizations," in *Proceedings of the 11th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. Citeseer, 2001.

[14] V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 2, no. 4, pp. 437–445, Dec. 1994.

[15] V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee, "Instruction Level Power Analysis and Optimization of Software," in *Proceedings of the 9th International Conference on VLSI Design: VLSI in Mobile Communication*, ser. VLSID '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 326–.

[16] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained Power Modeling for Smartphones Using System Call Tracing," in *Computer Systems, 2011. Proceedings., Sixth Conference on*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 153–168.

[17] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the Energy Spent Inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 29–42.

[18] M. Dong and L. Zhong, "Self-constructive High-rate System Energy Modeling for Battery-powered Mobile Systems," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 335–348.

[19] T. Li and L. K. John, "Run-time Modeling and Estimation of Operating System Power Consumption," in *Proceedings of the 2003 ACM SIGMETRICS*

International Conference on Measurement and Modeling of Computer Systems, ser. SIGMETRICS '03. New York, NY, USA: ACM, 2003, pp. 160–171.

[20] C. Sahin, F. Cayci, I. Gutierrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Initial Explorations on Design Pattern Energy Usage," in *Proceedings on the 1st International Workshop on Green and Sustainable Software (GREENS)*, June 2012, pp. 55–61.

[21] A. Litke, K. Zotos, A. Chatzigeorgiou, and G. Stephanides, "Energy consumption analysis of design patterns," in *Proceedings of the International Conference on Machine Learning and Software Engineering*. Citeseer, 2005, pp. 86–90.

[22] C. Seo, S. Malek, and N. Medvidovic, "An energy consumption framework for distributed java-based systems," in *Automated Software Engineering, 2007. Proceedings., Twenty-second International Conference on*, ser. ASE '07. New York, NY, USA: ACM, 2007, pp. 421–424.

[23] ——, "Component-Level Energy Consumption Estimation for Distributed Java-Based Software Systems," in *Component-Based Software Engineering*, ser. Lecture Notes in Computer Science, M. Chaudron, C. Szyperski, and R. Reussner, Eds. Springer Berlin Heidelberg, 2008, vol. 5282, pp. 97–113.

[24] A. Vetro, L. Ardito, G. Procaccianti, and M. Morisio, "Definition, Implementation and Validation of Energy Code Smells: an Exploratory Study on an Embedded System," in *Proceedings of the Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies(Energy)*, 2013, pp. 34–39.

[25] I. Manotas, C. Sahin, J. Clause, L. Pollock, and K. Winbladh, "Investigating the Impacts of Web Servers on Web Application Energy Usage," in *Proceedings of the 2nd International Workshop on Green and Sustainable Software(Greens)*. New York, NY, USA: ACM, 2013.

[26] L. Ardito, G. Procaccianti, M. Torchiano, and G. Migliore, "Profiling power consumption on mobile devices," in *Proceedings of the Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies(Energy)*, 2013, pp. 101–106.

[27] M. Dong and L. Zhong, "Chameleon: A Color-Adaptive Web Browser for Mobile OLED Displays," *IEEE Transactions on Mobile Computing*, vol. 11, no. 5, pp. 724–738, May 2012.

[28] L. Ding, T. Angelica, Huyen, and H. William, G.J., "Making Web Applications More Energy Efficient for OLED Smartphones," in *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, 2014.

[29] C.-L. Su, C.-Y. Tsui, and A. Despain, "Low Power Architecture Design and Compilation Techniques for High-performance Processors," in *Compcon Spring '94, Digest of Papers.*, Feb 1994, pp. 489–498.

[30] J. W. Davidson and S. Jinturkar, "Memory Access Coalescing: A Technique for Eliminating Redundant Memory Accesses," Charlottesville, VA, USA, Tech. Rep., 1993.

[31] H. Mehta, R. M. Owens, M. J. Irwin, R. Chen, and D. Ghosh, "Techniques for Low Energy Software," in *Proceedings of the 1997 International Symposium on Low Power Electronics and Design*, ser. ISLPED '97. New York, NY, USA: ACM, 1997, pp. 72–75.