An Investigation into the Energy Consumption of HTTP POST Request Methods for Android App Development

Hina Anwar, Dietmar Pfahl and Satish Srirama
Institute of Computer Science, University of Tartu, J. Liivi 2, Tartu 50490, Estonia

Keywords: Energy, HTTP-POST, Programming Patterns, Energy-profiling, Android Apps, Energy Consumption.

Abstract: Producing energy efficient applications without compromising performance is a difficult job for developers as it affects the utility of smart devices. In this paper, we conducted a small-scale evaluation of selected implementations using different methods for making HTTP POST requests. In the evaluation, we measured how much energy is consumed by each implementation and how varying message payload size effects the energy consumption of each implementation. Our results provide useful guidance for mobile app developers. In particular, we found that implementation using OkHttp consumes less energy than the implementation using HttpURLConnection or Volley libraries. These results serve to inform the developers about the energy consumption of different HTTP POST request methods.

1 INTRODUCTION

We are used to using different apps on smart devices that assist us in doing our day to day tasks or help us to connect with our peers. In order to carry out these tasks, smart devices communicate over the internet using WiFi, which is one of the most energy consuming tasks on these devices (Balasubramanian et al., 2009; Tawalbeh et al., 2016). As the battery capacity provided by these devices is limited, the extensive use of apps that frequently send and receive data to different servers over the network could drain the device's battery. Many advancements have been made in the hardware of smart devices making this communication more energy efficient but the hardware alone is not enough if the software designed for the hardware is not itself energy efficient (Anwar and Pfahl, 2017).

Producing an energy efficient app for smart devices is a difficult task as the implementation options vary widely in terms of energy consumption and performance. Therefore, the battery life of a smart device is one of the features that is used to evaluate the value of a device against its cost by technology reviewers. It has become a basic quality factor for selecting a device. Smart phone users frequently install apps from available online app stores and the rating of the apps are negatively affected if they consume too much energy and drain the device's battery (Li and Halfond, 2014). Therefore, app developers try to build apps that are energy efficient without compromising performance.

Research has been published in recent years related to energy code smells (Mannan et al., 2016; Carette et al., 2017), energy profiling of apps (Di Nucci et al., 2017) and energy saving programming practices (Hasan et al., 2016) in application development, with the intent to help android developers gain insights into the energy usage patterns of applications. While the energy consumption of HTTP GET requests have been analysed (Li and Halfond, 2014), analyses of HTTP POST requests have not yet been published.

In this paper, we present the results of a small-scale investigation into the energy consumption of different methods for HTTP POST request and how changing the message payload size effects the energy consumption of these methods. As apps nowadays are designed to frequently send and receive data to and from servers, we wanted to estimate the energy consumption of the various methods for sending data to the server.

The results of this investigation could be informative, especially for beginner level app developers, to better understand the energy behaviour of their apps. Our experimental results show that for making an HTTP POST request to send a string, the implementation using the OkHttp consumes the least amount of energy. While the alternative

Anwar, H., Pfahl, D. and Srirama, S.
An Investigation into the Energy Consumption of HTTP POST Request Methods for Android App Development.
DOI: 10.5220/0006846102410248
In *Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018)*, pages 241-248
ISBN: 978-989-758-320-9

implementations to make an HTTP POST request using HttpURLConnection which is included in android SDK and Google Volley libraries consume slightly more energy. We also used different message payload sizes ranging from 10 bytes to 10,000 bytes to investigate how message payload size effect the energy consumption of each implementation. The experimental results show that for each size OkHttp consumes the least amount of energy. Also, the results indicated that implementation using OkHttp is faster than the implementations using Volley and HttpURLConnection for making HTTP POST request.

The remainder of the paper is organized as follows. In Section 2 we present the related work. Section 3 provides details of research method, hypothesis and the experimental setup. Section 4 presents the results. Section 5 presents a discussion of the results. Section 6 presents threats to validity. Finally, we summarize our findings and discuss future directions for research in Section 7.

2 RELATED WORK

Empirical research has been done for optimizing the performance of android based devices. There is one group of publications which focuses on energy impact of code smells in android (Mannan et al., 2016; Carette et al., 2017; Fowler et al., 1999). Another group focuses on energy based profiling of android apps (Metri et al., 2015; Nucci et al., 2017). While another group of studies focuses on energy consumption of HTTP request and define approaches to bundle them in order to reduce energy (Li and Halfond, 2014; Li and Halfond, 2015; Li et al., 2016).

All the related work mentioned above provide information regarding energy consumption at different levels, e.g., source code level, methods level. However, the literature does not provide information to the developers regarding energy consumption of different network libraries. In our study, we present the energy consumption of HTTP POST requests using different methods to see how much energy they consume, in order to better inform the application developers.

The closest work to our paper is the work done by Li and Halfond (2014) in which among other things they have measured the energy consumption of HTTP GET request method for different file sizes using HttpURLConnection. In our study, we compare the energy consumption of different HTTP POST request methods for different message payload sizes.

3 RESEARCH METHOD

In this section, we describe the experiment we undertook to evaluate the energy consumed by different HTTP POST request methods. The experiment addresses the following research question.

RQ1: What is the energy consumption of different HTTP POST request methods?

RQ2: What is the effect of message payload size on the energy consumption of different HTTP POST request methods?

RQ1 compares the energy consumption of different method implementations for making HTTP POST request, each using a different network library. RQ2 compares the energy consumption of different method implementations for sending different message payload sizes to the server.

3.1 Experiment Design

The official android developer blog[1] suggests libraries and classes for network operations based on two factors: simplicity to implement and performance. We did not see any discussions or evidence about the energy consumption of these libraries.

The new android profiler replaces the android monitor tool and provides a new suite of tools to measure an app's CPU, memory, and network usage in real-time. Currently, the network profiler works with HttpURLConnection, OkHttp and Volley network libraries (Android Developers n.d.; Jamal 2017). Therefore, in our experiment, we include the method using HttpURLConnection and compare its energy consumption to the methods using the OkHttp and Volley libraries for making an HTTP POST request.

Many other libraries are also available, e.g., Retrofit, HttpClient and others, for making HTTP POST requests, but they are either not supported by the new android profiler for profiling network operations in android studio 3.0 or the libraries are deprecated. The libraries are selected based on the fact that they are maintained by android and supported by android profiler in android studio 3.0.

[1] https://android-developers.googleblog.com/

Table 1: Selected Methods.

	Description
Method 1	Method 1 uses HttpURLConnection, which is derived from "URLConnection with support for HTTP-specific features"(Android Developers n.d.).
Method 2	Method 2 uses OkHttp, which is a library developed by square and based on Okio library and it supports networking for android 2.3 and above (Square Inc. n.d.).
Method 3	Method 3 used Volley, which is an HTTP library that makes networking for android apps easier (Android Developers n.d.).

To estimate the energy consumption of different implementations of HTTP POST requests, a basic android app with a blank screen was created which makes HTTP POST requests and logs the server response. Once the HTTP request is complete the app terminates automatically. This android app adheres to the REST architectural style. Android settings for screen timeout, display brightness and sound profiles are kept constant throughout this experiment. We wrote three different versions of the app, each using a different method from Table 1. We refer to each version of the app as a unique implementation of the HTTP POST request. Each implementation is performing asynchronous tasks. In each implementation, the code for HTTP POST request is put inside a loop with N=1000, and it is sending a string to the server and getting the server response. Sending the string in a loop and getting the server response is counted as one run. Before we execute an implementation, we wait ten seconds for the Arduino connection to settle down and to sync with the NTP server.

In the experiment, we first measure the energy consumed by the mobile device in an idle state to establish a baseline. This baseline is subtracted from the readings to filter out the energy consumed by HTTP Post request methods. We execute each run five times to account for the underlying variation in the mobile device and then average the collected data for the final readings.

We use the time stamps from the adb (android debug bridge) logs to determine when the execution of a given implementation starts and completes. In adb log, we record the time stamp when each post request is completed. The time difference between completion of first and last post request represents the execution time of the implementation. Energy values are expressed in milliampere-hour (mAh).

We varied the message payload size from 10 to 10,000 bytes (11 different sizes). For each size, we repeated the experiment for each of the three implementations as explained above (total runs= 11 different message payload sizes x 5 runs for each size x 3 different implementations).

3.2 HTTP Request Implementations

In this section, we describe the implementations for making HTTP POST request using different methods.

3.2.1 Implementation 1

This implementation uses Method 1 as described in table 1. In Figure 1, lines 1 to 3 contains infrastructure code to setup the task for android. In lines 4 to 6, we execute AsyncTask using execute() inside a loop with N=1000. In Line 9 to 21 is the AsyncTask which enables proper use of UI thread. 8 to 20 we execute the doInBackground() method which contains the task to be performed on a separate thread. The server response is logged in line 19. The server response, in this case, is the server status code, i.e. 200, converted to a string.

3.2.2 Implementation 2

This implementation uses Method 2 as described in table 1. In Figure 2, lines 1 to 3 contains infrastructure code to setup the task for android. In lines 4 to 9, we instantiate an OkHttp client, set the data format, create the request object (containing the string) to be posted, and create the request for the server. In line 10 to 18, we execute this request inside a loop with N=1000. We make an asynchronous network call using the enqueue() method with call object and pass the anonymous call-back object to onFailure() and onResponse() methods. In line 17 the server response is logged. The server response, in this case, is the server status code, i.e. 200, converted to a string.

3.2.3 Implementation 3

This implementation uses Method 3 as described in table 1. In Figure 3, lines 1 to 3 contains infrastructure code to setup the task for android. In line 4 we instantiate the request queue. In line 6 we create the string request that we will assign to the queue we created earlier. The string request makes a post request, listens for response and handles errors if any. In line 8 to 9, we set the data format for a request. At line 11 to 14, we set the string to send. In line 16 to 18, we log the server response, i.e. the status code. In line 19 to 21, we run the loop with N=1000 and add the request to the queue.

```
public void onCreate(Bundle savedInstanceState)
{super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
for (int i = 0; i <= 1000; i++) {
new FirstImplementationTask().execute();}
finish(); }
class FirstImplementationTask extends AsyncTask
<String, Void, String> {
protected String doInBackground(String... urls)
{try {
URL url  = new URL("resourcelink");
HttpURLConnection urlConnection =
(HttpURLConnection)
url.openConnection();
urlConnection.setDoOutput(true);
urlConnection.setRequestMethod("POST");
OutputStreamWriter wr = new OutputStreamWriter
    (urlConnection.getOutputStream());
wr.write("This is ap");
wr.flush();
Stringtext=urlConnection.getResponseMessage();
Log.i(tag: "response", msg "" +text); }
urlConnection.disconnect(); }
return null;  }}}
```

Figure 1: Pseudocode for implementation 1 of the HTTP POST request using Method 1.

```
public void onCreate(Bundle savedInstanceState)
{super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
OkHttpClient client = new OkHttpClient();
MediaType textPlainMT = MediaType.parse
("text/plain; charset=utf-8");
String value = "This is ap";
RequestBody body = RequestBody.create
 (textPlainMT, value);
Request request = new Request.Builder().url
("resourcelink").post(body).build();
Response response = null;
for (int i = 0; i <= 1000; i++) {
client.newCall(request).enqueue(new Callback(){
@Override
public void onFailure(Call call, IOException e)
{..}  @Override
public void onResponse(Call call,
final Response response)
final String responseData=response.body()
.string();
Log.i(tag: "response", msg "" + responseData);
}});}finish();}
```

Figure 2: Pseudocode for implementation 2 of the HTTP POST request using Method 2.

```
protected void onCreate(Bundle savedInstanceS
{super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
RequestQueue queue = Volley.newRequestQueue
(MainActivity.this);
final String url = "resourcelink";
StringRequest req = new StringRequest
(Request.Method.POST,url, new
Response.Listener<String>(){…},
new Response.ErrorListener(){..}){
@Override
public String getBodyContentType() {
return "application/json; charset=utf-8";}
@Override
protected Map<String,String> getParams(){
Map<String,String> params = new HashMap
<String, String>();
params.put("txt","This is ap");
return params; }
@Override
protected Response<String> parseNetworkResp
(NetworkResponse response) {
int mStatusCode = response.statusCode;
Log.i(tag: "response", msg "" +text);…..}};
for (int i = 0; i <= 1000; i++) {
queue.add(req);}
finish(); }
```

Figure 3: Pseudocode for implementation 3 of the HTTP POST request using Method 3.

3.3 Energy Measurement

Our experiment was performed on an LG Spirit running android 6.0. For network access, the android device and server were connected to the same WiFi network. The energy consumption on the phone was measured by a setup consisting of an Arduino Mega ADK and INA219 current sensor. The Arduino board collects the energy measurement via an INA219 current sensor that internally averages readings to produce approximately 128 current readings per second. The INA219 chip reports current and voltage measurements over 12C to the Arduino. The readings from Arduino Mega ADK were stored on a desktop computer using PySerial.

For controlling the charging of the mobile device during the experiment a TIP127 transistor was used in the circuit. By default, the mobile device is connected but when the signal is sent by Arduino the transistor will block the USB charging connection for the phone during the experiment. The above measurement setup is inspired by the work of Hindle et al., (2014). Arduino Ethernet Shield is used on top of the Arduino board to get time stamps synced with NTP server for each current reading.

4 RESULTS

In this section, we present the experimental results for each research question.

RQ1: What is the energy consumption of different HTTP POST request methods?

Table 2 presents the total energy consumption of each implementation for message payload size 10 bytes. Implementation 1 takes on average 44.9 sec to complete 1000 HTTP POST requests and the energy consumed is 0.376857 mAh. Implementation 2 takes on average 10.2 sec to complete 1000 HTTP POST requests and the energy consumed by implementation 2 is 0.112997 mAh. Implementation 3 takes on average 15.8 sec to complete 1000 HTTP POST requests and the energy consumed by implementation 3 is 0.392877 mAh. Implementation 2 is using approximately 70% less energy than implementation 1 and approximately 71% less energy than the implementation 3. Time consumed by implementation 2 to complete the 1000 POST requests is approximately 77% and 35% less than the implementation 1 and 3 respectively.

Table 2: Average energy consumption and average completion time of each implementation for 5 runs.

	E(mAh)	Time(sec)
Implementation 1	0.376857	44.9
Implementation 2	0.112997	10.2
Implementation 3	0.392877	15.8

RQ2: What is the effect of message payload size on the energy consumption of different HTTP POST request methods?

Figure 5 shows the energy consumed by the HTTP POST request using HttpURLConnection, OkHttp and Volley libraries, respectively for different message payload sizes. The y-axis of figure 5 shows energy consumption in mAh. The x-axis in figure 5 is the message payload size in bytes.

We can see from the figure 5 that the energy consumption of each implementation is different at ach point. For Implementation 1 (represented by circles in figure 5) the energy consumption is in the range of 0.3-0.4 mAh when the message payload size is less than 500 bytes. For message payload size greater than 500 bytes energy consumption starts to increase.

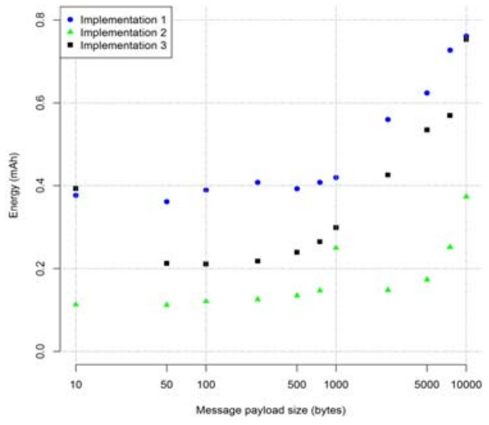

Figure 4: Energy consumption of Implementation 1, 2 and 3 for posting message payloads of different sizes.

For Implementation 2 (represented by triangles in figure 5) the energy consumption is in the range of 0.1-0.2 mAh when the message payload size is less than 500 bytes. For message payload size greater than 500 bytes energy consumption starts to increase, however, it is still significantly lower than Implementation 1. For Implementation 3 (represented by squares in figure 5) the energy consumption is in the range of 0.2-0.3 mAh when the message payload size is less than 500 bytes. For message payload size greater than 500 bytes energy consumption starts to increase but it is lower than Implementation 1 and higher than Implementation 2.

5 DISCUSSION

In online forums[2], android developers discuss why they prefer one implementation over the other based on their experience. Usually, the recommendations are based on scenario based performance. However, we have not yet seen any published results about which implementation consumes less energy for making HTTP POST request.

From the result section above we have seen that the implementation using OkHttp uses relatively less energy, for making HTTP POST request for sending a string to the server, as compared to the implementation using HttpURLConnection and Volley. This investigation was not intended to compare the purpose and design of each library, however, based on the results we can make some observations about the energy consumption of HTTP POST request operation of these libraries that might be useful for android developers.

[2] https://stackoverflow.com/
https://android-developers.googleblog.com/

In our experiment, implementation 1 and 3 take more time to complete and consume more energy, while implementation 2 consumes the least amount of energy and time.

Table 3: Energy consumption and completion time for different message payload sizes for Implementation 1, 2 and 3.

Message payload size (bytes)	Implementation					
	1		2		3	
	Energy (mAh)	Time (sec)	Energy (mAh)	Time (sec)	Energy (mAh)	Time (sec)
10	0.37	44.9	0.11	10.3	0.39	15.9
50	0.36	38.4	0.11	8.0	0.21	17.7
100	0.38	41.8	0.12	8.4	0.21	18.9
250	0.40	43.0	0.12	9.4	0.22	19.3
500	0.39	40.4	0.13	8.0	0.24	22.3
750	0.40	48.0	0.15	9.0	0.26	22.9
1000	0.41	45.1	0.25	10.4	0.30	26.1
2500	0.55	51.2	0.15	11.4	0.42	31.9
5000	0.62	47.2	0.17	11.8	0.53	37.9
7500	0.72	59.2	0.25	13.2	0.57	44.0
10000	0.76	62.5	0.37	19.3	0.75	51.8

We have considered the effect of varying message payload size on the energy consumption of each implementation. From the work of Li and Halfond (2014), we know that for GET requests (using HttpURLConnection) the relationship between the size of downloaded file and energy consumption is linear when the data size is greater than 1024 bytes. Compared to the results of Li and Halfond the results of our experiment indicate that the HTTP POST request using implementation 1 (i.e. HttpURLConnection) use less energy than the GET request method.

However, we cannot generalize based on these results as the research method in both experiments might not be identical, i.e. in our experiment we are using asynchronous tasks. Also, the paper by Li et al. does not discuss implementation 2 and 3.

Table 4: Comparison of HTTP GET (Li and Halfond, 2014) and HTTP POST request for Method 1.

Message payload size	Energy consumed by GET request	Energy Consumed by POST request
10- 1000 bytes	0.5-0.6 mAh	0.3-0.4 mAh
1000-10000 bytes	0.6-1.1 mAh	0.4-0.76 mAh

In HTTP requests "energy consumption is dominated by transmitting protocol and control information when the packet size is small" [4]. Therefore, for making multiple HTTP POST requests containing at least 10-10000 bytes of data the implementation with OkHttp could be used as it consumes less energy than the implementations with Volley and HttpURLConnection. The difference in energy consumption of each implementation for message payload sizes less than 500 bytes is not significant as discussed in the results. However, once the size of message payload exceeds 500 bytes the increase in energy consumption is more observable. We did not find any linear relationship between completion time and energy consumption of each implementation, therefore from the results, we can conclude that more time does not mean more energy. However this behaviour is true for our experiment only, we do not claim it to be generally true as more experimentation is needed to generalize it. For each message payload size, completion time and energy consumed by implementation 2 are lower than the implementation 1 and 3.

In our experiment the mobile device was already connected to the WiFi network before we started the measurements, therefore we could ignore the energy consumed by scan and association overhead of WiFi. "In WiFi, the data transfer itself is significantly more energy efficient than 3G for all transfer sizes" (Balasubramanian et al., 2009). As we are posting strings less than 1MB in size and the transfers are frequent and successive, we can deduce that the energy consumption over 3G could be more than WiFi. However this need to be tested through experimentation.

In our experiment implementations were performing asynchronous tasks i.e. multiple threads were created to make HTTP POST requests. We intentionally set N=1000 in all implementations in order to make the energy consumed by HTTP POST requests observable. However, the thread management in each of the implementation is different due to different network libraries (discussion on thread management is out of the scope for this paper) and increasing the value of N in the loop could cause potential memory heap problem at which point garbage collector would kick in. One way of avoiding the potential memory heap problem is to use the built-in tools[3] available in the android studio like memory monitor. For more specific details about memory usage, the allocation tracker tab or the heap tab available in the android studio could be useful.

[3] More information about these tool could be found on: https://developer.android.com

From adb logs, we know that during the execution of HTTP POST request other unavoidable background operations (such as inter-process communication between kernel module and android service or Network/connectivity events to check if the WiFi connection is still available or connected) were also performed by android OS. These events are happening concurrently with the app and are quite normal in any android environment.

6 THREATS TO VALIDITY

A number of issues affect the validity of this work. The comparison done in this study is based on the most basic operation offered by the selected libraries, i.e., for posting a string in a RESTful environment. The result does not claim to generalize the behaviour of these libraries in terms of energy consumption. Neither do we discuss any other networking operation offered by these libraries. The discussion about complexity and use of these selected libraries performing an operation other than the HTTP POST request is out of the scope of this paper. We do not claim the methods evaluated in this paper to be the most commonly used for performing HTTP POST requests, but they are the methods that appear most frequently in google searches when phrases similar to "http post request for android applications" are used.

Experimental results presented in this paper were performed on a single android device using a specific android version, but they could differ slightly on other devices. Any measurements taken using real mobile devices are always prone to unwanted noise and overheads. In order to minimize this effect we placed the code inside the onCreate() method to ensure a fixed overhead. To ensure that the non-deterministic noise is filtered out we first calculated a baseline, then took measurements five times and filtered out the baseline from the readings to cancel out the noise. As the energy consumption of a single HTTP POST request might not be large enough to be observable we placed the HTTP POST request code inside a loop with N=1000. This ensured that the changes were observable. We recorded adb logs for each run using consistent code in each implementation. Therefore any overhead introduced by logging was consistent in all implementations. As we are interested in the comparison of energy consumption of the selected implementations, the consistent overhead due to logging was ignored as it could not have affected the results. In addition, WiFi networks are subject to latencies and delays caused due to equipment, processing, transmission, queuing and propagation. Therefore energy measurement and completion time of the HTTP POST requests could be different on different networks.

7 CONCLUSION AND FUTURE WORK

Producing energy efficient applications without compromising on performance is a difficult job for developers as it affects the utility of smart devices. Recent research in the domain of android development tend to be oriented towards energy code smells and energy profiling of apps but there exists a gap in this literature regarding the energy consumption of different implementations for making HTTP POST requests. The resulting situation is that developers lack information about the energy consumed by different implementations for making HTTP POST request. In this paper, we conducted a small-scale evaluation of different implementation using selected methods for making HTTP POST requests. In the evaluation, we assessed how much energy is consumed by each implementation and how changing the message payload size effects the energy consumption of these implementations. Our results provide useful guidance for mobile app developers. In particular, we found that implementation using OkHttp consumes less energy than the implementations using HttpURLConnection or Volley libraries. The results also indicated that HTTP POST request implementations using OkHttp takes less time to complete. These results serve to inform the developers about the energy consumption of different HTTP POST request methods.

Future work involves the extension of this work to include more implementations using a variation of methods. We also plan to investigate the energy comparison of HTTP POST vs. HTTP GET when different data formats and networks (such as 3G/4G/GSM) are used.

ACKNOWLEDGEMENT

This work is supported by the institutional research grant IUT20- 55 of the Estonian Research Council and the Estonian Center of Excellence in ICT research (EXCITE). The authors would like to thank Alo Peets for helping with the experimental setup.

REFERENCES

Android Developers, HttpURLConnection. Available at: https://developer.android.com/reference/java/net/Http URLConnection.html [Accessed December 30, 2017a].

Android Developers, Inspect Network Traffic with Network Profiler. Available at: https://developer. android.com/studio/profile/network-profiler.html [Accessed December 30, 2017b].

Android Developers, Transmitting Network Data Using Volley. Available at: https://developer.android.com/ training/volley/index.html [Accessed December 30, 2017c].

Anwar, H. & Pfahl, D., 2017. Towards Greener Software Engineering Using Software Analytics: A Systematic Mapping. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE*, pp. 157–166.

Balasubramanian, N., Balasubramanian, A. & Venkataramani, A., 2009. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference - IMC '09*. New York, New York, USA: ACM Press, p. 280.

Carette, A., Younes, M.A.A., et al., 2017. Investigating the energy impact of Android smells. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE*, pp. 115–126.

Carette, A., Adel, M., et al., 2017. Investigating the Energy Impact of Android Smells. pp.115–126.

Fowler, M. et al., 1999. Refactoring: Improving the Design of Existing Code, *Reading*, MA: Addison-Wesley.

Hasan, S. et al., 2016. Energy profiles of Java collections classes. In P*roceedings of the 38th International Conference on Software Engineering - ICSE '16*. New York, New York, USA: ACM Press, pp. 225–236.

Hindle, A. et al., 2014. GreenMiner: a hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*. New York, New York, USA: ACM Press, pp. 12–21.

Jamal, E., 2017. Android Developers Blog: Android Studio 3.0 Canary 1. Available at: https://android-developers.googleblog.com/2017/05/android-studio-3-0-canary1.html [Accessed December 30, 2017].

Li, D. et al., 2016. Automated energy optimization of HTTP requests for mobile applications. In *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*. New York, New York, USA: ACM Press, pp. 249–260.

Li, D. & Halfond, W.G.J., 2014. An investigation into energy-saving programming practices for Android smartphone app development. *Proceedings of the 3rd International Workshop on Green and Sustainable Software - GREENS 2014*, pp.46–53.

Li, D. & Halfond, W.G.J., 2015. Optimizing energy of HTTP requests in Android applications. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile - DeMobile 2015*. New York, New York, USA: ACM Press, pp. 25–28.

Mannan, U.A. et al., 2016. Understanding code smells in Android applications. In *Proceedings of the International Workshop on Mobile Software Engineering and Systems - MOBILESoft '16*. New York, New York, USA: ACM Press, pp. 225–234.

Metri, G., Shi, W. & Brockmeyer, M., 2015. Energy-Efficiency Comparison of Mobile Platforms and Applications: A Quantitative Approach. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications. HotMobile '15*. New York, NY, USA: ACM, pp. 39–44.

Nucci, D. Di et al., 2017. Software-Based Energy Profiling of Android Apps : Simple, Efficient and Reliable ? , pp.103–114.

Di Nucci, D. et al., 2017. Software-based energy profiling of Android apps: Simple, efficient and reliable? In 2*017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE*, pp. 103–114.

Square Inc., n., An HTTP & HTTP/2 client for Android and Java applications. Available at: http://square.github.io/ okhttp/ [Accessed December 30, 2017].

Tawalbeh, M., Eardley, A. & Tawalbeh, L., 2016. Studying the Energy Consumption in Mobile Devices. *Procedia Computer Science, 94*, pp.183–189.