EasyChair Preprint

№ 10599

Identifying Sources of Energy Consumption for Android Applications: a Pilot Study

Anshu Rathor, David Berdik, Fadi Wedyan and Yaser Jararweh

July 20, 2023

Identifying Sources of Energy consumption for Android Applications: A Pilot Study

Anshu Rathor*, David Berdik†, Fadi Wedyan‡, Yaser Jararweh§

Duquesne University, Pittsburgh, Pennsylvania, USA
*rathora@duq.edu, †dgberdik@gmail.com, §jararwehy@duq.edu
Lewis University, Romeoville, Illinois, USA
‡fwedyan@lewisu.edu

Abstract—Over the years, the capabilities of cell phones have expanded far beyond their intended core functionality of making and receiving calls. Modern smartphones are effectively pocket-sized computers with a built-in display, CPU, GPU, memory, storage space, and so on. Although these advancements in mobile technology are impressive, they are unfortunately limited by a constraint that is not present with traditional computers: lack of a consistent power source. To work around this limitation, mobile operating systems apply optimizations to resource management that are not used with traditional operating systems, and application developers must take these limitations in to consideration as well. The need to optimize this process has given rise to the development of tools to analyze and identify how and when power is used. In this paper, we present and compare various tools aimed at performing this type of analysis on Android devices.

I. Introduction

Over the years, the capabilities of cell phones have expanded far beyond their intended core functionality of making and receiving calls. Modern smartphones are effectively pocket-sized computers with a built-in display, CPU, GPU, memory, storage space, and so on. Although these advancements in mobile technology are impressive, they are unfortunately limited by a constraint that is not present with traditional computers: lack of a consistent power source [1]. To work around this limitation, mobile operating systems apply optimizations to resource management that are not used with traditional operating systems, and application developers must take these limitations in to consideration as well. The need to optimize this process has given rise to the development of tools to analyze and identify how and when power is used. In this paper, we present and compare various tools aimed at performing this type of analysis on Android devices [2].

In order to understand power usage estimation, we must first understand the fundamentals of what "power" and "energy" refer to in this context as well as what the differences are between the two. Within this context, "power" refers to the pace at which work is done and is estimated in watts. Energy, on the other hand, is the amount of power utilized over the long run and is communicated in joules. For example, a process that utilizes 5 watts of power for 30 seconds is said to have exhausted 150 joules of energy. Although power and energy are relative, utilizing less power for a given process does not imply that less energy is used in the long run. For instance, an application may have a higher power usage than another application with similar functionality, yet it may use substantially less energy if the overall runtime of the application is lower [3].

II. Measuring Power Consumption

Estimating power usage of smartphones is a challenging task, particularly among Android devices, because of the extremely diverse collection of available devices. Additionally, independent researchers' ability to identify power consumption patterns of a given smartphone application or hardware component can be hindered by lack of access to proprietary source code or hardware information. To work around this, researchers must resort to reverse-engineering.

Power can be estimated at two levels of granularity: coarse-grained (at the application level) and fine-grained (at the capacity level or guidance level). Coarse-grained estimations give the general power utilization of the application to help the client and analyzer of an application and fine-grained will give actual data to investigate an application at improvement time [4].

Various tools are available for measuring power usage, and can provide various types of data depending on the needs of the user. These tools can either be hardware-based or software-based. In the following two subsections, we discuss both types of tools.

A. Hardware-Based Measuring Tools

Hardware-based power measuring tools are very useful for obtaining the most accurate and precise result, however, these tools have several drawbacks. Specifically, hardware-based tools tend to be very expensive, and even if finances are not a limiting factor, the usage of these tools requires an understanding of how to use them properly in order to obtain an accurate result.

An example of one such external device is the Monsoon Power meter, which can be used to measure the current drawn from the battery by the cell phone [3]. Another

approach to utilizing an external hardware-based tool is to mount a voltage meter over a resistor associated in series with the cell phone's power supply, enabling the current to be measured from a voltage change. An external battery or other external power supply can be used when measuring power utilization. If price and knowledge of proper usage are not limiting factors, hardware-based power measuring tools are preferable to software-based tools thanks to their ability to provide extremely precise data. In upcoming sections of the paper, we will discuss hardware-based tools further, as they are generally used to provide ground truth values for software-based tools.

B. Software-Based Measuring Tools

Software-based power measuring tools operate as applications installed on the device. These tools operate by relying on numerical models that are created using hardware-based measuring tools such as those described previously. Various components of the device such as the CPU, memory, GPS, and other sensors, each have a unique model associated with them, which is used by the application to generate the overall usage reports. Application-based solutions for measuring power usage have the advantage of being cheaper and easier to use than their hardware-based counterparts, but are unfortunately believed to be less accurate [5]. Determining an ideal model to use for software-based solutions can be done in two different ways. The fist method involves using hardware-based tools to determine how much power is used by various components of the device under different circumstances and the second method involves adding power usage monitoring to the firmware of the various components in the device.

III. Battery-Draining Components of a Smartphone

In this section, we will describe the components of a smartphone that contribute to battery drainage. In a 2010 study published by Aaron Carroll and Gernot Heiser, the power usage of various critical points of a smartphone, such as the GPS, were measured. As part of the testing, each component was subjected to various use scenarios in order to determine how it impacted battery drainage. As would be expected, the power drainage of each component varied depending on usage. For example, calls drained 834mW and GPS, 143-166mW on average. Screen backlight amounts to between 7-8mW to 404mW depending on brightness. It should also be noted that the testing device is a 2.5G phone, and 3G drains more power [6].

Smartphone components such as accelerometers, cameras, GPS, compasses, gyroscopes, gravity sensors, light sensors, proximity sensors, pressure sensors, network radios, flash memory, RAM, CPUs, and GPUs are also an additional source of power drainage for a smartphone. Figure 1 shows the significant parts of the cell phone that use battery power. In light of the review via Carroll and Heiser, the power usages of cell phone parts are talked about beneath.

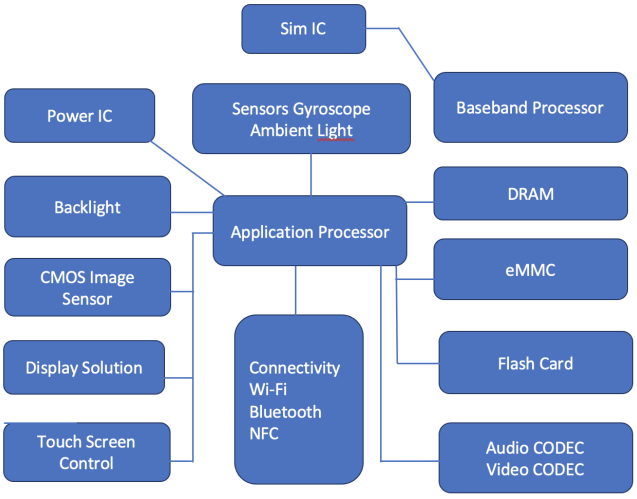

Fig. 1. Power consuming components of Smartphone

A. Screen

The screen light took power over the scope of a whole number worth between 1 and 255. The power management module constrains this. An ordinary Brightness control UI gave the value somewhere in the range of 30 and 255. The base backlight power is approximately7.8m W, and the maximum is 414mW. The backdrop illumination devours negligible power when incapacitated.

B. CPU and RAM

On average, a smartphone's CPU and RAM account for about 15% of device usage, however, that number can rise higher when they are under heavy load such as when the user is streaming videos or browsing the internet. About 7% to 10% of smartphone battery usage can be accounted for by read and write operations performed to the phone's internal storage.

C. Network

Wi-Fi and GPRS are prominent supporters of the use of force. Wi-Fi showed a throughput of 660.1 $\pm$ 36.8 KiB/s and GPRS 3.8 $\pm$ 1.0 KiB/s. The expanded CPU and RAM power for Wi-Fi mirrors the expense of handling information with higher throughput. The impact of sign strength on power brought about an expansion of GSM power of30-percent, yet no impact on throughput.

D. GPS

Table I shows battery usage of the GPS component of a smartphone under three different circumstances.

State	Power(mW)
Enabled (Internal Antenna)	143.1 ± 0.05%
Enabled (External Antenna)	166.1 ± 0.04%
Disabled	0.0

TABLE I
GPS Energy Consumption

E. Bluetooth

To measure Bluetooth power usage, audio was played to a Bluetooth headset. To determine the power usage of just the Bluetooth radio without factoring in the power draw from playing audio, the power draw from just playing audio was measured and subtracted from the measured draw from using a Bluetooth headset. The headset was placed approximately 30cm from the phone for the close benchmark and 10m in the far benchmark, as shown in table II.

Benchmark	Power (mW)	Power (mW)
-	Total	Bluetooth
Audio baseline	459.7	-
Bluetooth (near)	495.7	36.0
Bluetooth (far)	504.7	44.9

TABLE II
Bluetooth power under the audio benchmark

F. Device Usage Scenarios

1) Audio Playback: A smartphone's audio subsystem typically consumes 33.1mW. Approximately 58% of this energy is consumed by the codec, with the remaining 42% used by the amplifier. Overall, the audio subsystem accounts for less than 12% of energy consumed. The additional energy consumed in the high-volume benchmark is more minor than 1mW compared with the low-volume case.

2) Video Playback: To test battery usage when playing back video, a 5 minute video without sound was played at varying levels of screen brightness. The brilliance levels with backdrop illumination power on were 30, 105, 180, and 255. GSM power was remembered for the estimations. The CPU is the greatest single benefactor of force. The showcase subsystems represent 38% of total power, up to 68% with the most extreme backdrop illumination splendor. Immaterial power needs to stack the video from the SD card.

3) Phone Calls: Estimating force usage of a GSM call incorporates stacking the dialer application, dialing a number, and settling on a 57-second decision. Along these lines the time spent in the call was roughly 40 seconds, expecting a 7-second association time. The complete benchmark runs for 77 seconds. GSM power rules in this benchmark at 832:4 ± 99:0 mW. Android cripples the backdrop illumination during the call.

4) Web Browsing: The power utilization for web-browsing workload using both GPRS and Wi-Fi connections. The benchmark ran for 490 seconds which consisted of loading the browser application, selecting a website, and browsing several pages.

5) Camera Usage: The camera is quite possibly the main bit of equipment on a smartphone. Notwithstanding, it additionally can deplete the battery a lot. The first and most apparent explanation is that it is a different piece of components. Although, by far, most of the camera battery channel comes from screen and processor utilization. Your presentation is required as a viewfinder, and some OEMs even knock up the brilliance of the display when in camera mode. Moreover, every advanced smartphone probably has some post-processing, and that likewise requires additional handling power.

G. Causes of Battery Drainage

In addition to battery drainage caused by the previously-discussed usage scenarios, software bugs can also cause significant battery drainage. The following subsections describe some examples of battery-draining bugs.

1) Missing Sensor Deactivation Bugs: To use a sensor, an application needs to register a listener with OS and specify a sensing rate [9]. When the use of sensors is finished, its listener should be unregistered in time. Forgetting to un-register sensor listeners can cause unnecessary sensing operations.

2) Wake Lock Registration Bugs: Some smartphone applications will establish a so-called "wake lock" that instructs the operating system to keep the device's screen on and not allow the device to enter a low-power state. This is useful for applications that need access to system resources for an extended period of time, however, an application's failure to release a wake lock after it is no longer needed can cause the device's battery to drain at a faster rate.

3) Unnecessary Sensor Utilization: Any use of the device's sensors results in battery drainage, however, this is acceptable if the use of the sensors provides meaningful benefits to the user. For cases in which an application activates a device sensor but does not meaningfully use the data that it acquires from the sensor, power is effectively being wasted by the application in question.

H. Causes from Event-driven Programming

In programming, Event-driven computer programs are programming in which the program's progression is dictated by events, for example, client activities (mouse clicks, key presses), sensor yields, or message passing from different projects or threads. Event-driven computer programs are prevailing in graphical UIs and different applications (e.g., JavaScript web applications) that focus on playing out specific activities in light of client input.[7] This is likewise valid for programming for smartphones drivers.

An ordinary client confronting smartphone application is composed of many events controllers with events being

a client or outer exercises.[7] The programmer needs to monitor every conceivable event and when it is triggered and manipulate the wake locks likewise.

The Dialer app in smartphones applications executes the dialing function of the smartphone. The application is set off when the client gets an approaching call or taps the telephone symbol to settle on an outgoing call. To execute this call or function, the application unequivocally primary tains three wake locks: FULL-WAKE-LOCK for keeping the screen on (e.g., in the circumstances like when the client is dialing the numbers to call), PARTIAL WAKE LOCK for keeping the CPU on (e.g., if there should arise an occurrence of an approaching consider when the telephone is turned off), and PROXIMITY SCREEN OFF WAKE LOCK which turns the vicinity sensor on and off (to recognize client's closeness to the telephone).

To deal with the three wake locks, the application expressly keeps a state machine where the states address the lock conduct, i.e., which lock should be obtained and delivered, and the "condition" of the telephone addresses the state advances. The conditions are different and incorporate occasions, for example, (a) if the telephone gets a call, (b) if the telephone is squeezed against the client's ear wherein case the closeness sensor triggers the screen to go off, (c) if the call closes, (d) if a wired or Bluetooth headset is connected (e.g., in a call), (e) if the telephone speaker is turned on, (f) if the telephone slider is opened in the middle of calls, and (g) if the client clicked home caught in a call. For every one of these setting off occasions, the telephone changes the condition of the wake lock state machine, gaining one and delivering another.

IV. Examples of Software-Based Measuring Tools

A. PETrA

PETrA is a desktop application written in Java that can be used by Android app developers to measure power usage of their application when it is executed in various circumstances. It is compatible with Android 5 (Lollipop) and higher and relies on power usage models to generate its results [8].

B. Android Profiler

Google's official tool for application power usage is the the Android Profiler, which was initially released with Android Studio 3.0 and replaced the older Android Monitor tools. The purpose of the tool is to provide developers with information about how their app uses a device's CPU, memory, network, and battery. The energy profiler appears as a row in the profiler window when you run your app on a connected device or Android Emulator running Android 8.0 (Oreo) or higher.

C. Test Environment Setup for PETrA and Android Profiler

- Selected applications from different criteria
- Download those applications
- Run the selected Android applications on the selected tool(Petra/Android Profiler)
- Analyzing the measurements
- Report the results of applications

ID	Name	Category
1	Newson	News/Magazine
2	Notepad	Tool
3	Oxford Dictionary	Tool
4	Angry Bird	Game
5	YouTube	Video
6	Bubble Blast	Game
7	Sniper Shooter	Game
8	Accu Battery	Tool
9	Google Map	Navigation Tool
10	Calculator	Tool

TABLE III
Applications table and their category

PETRA shows the configuration view. Using this window, the developer can customize PETRA concerning the location of the apk file, the ANDROID MONKEY options, the ANDROID MONKEYRUNNER script location, the number of times (i.e., runs) the energy measurements must be computed, the location of the ANDROID SDK, and the location of the XML file containing the power profile of the device. Regarding the ANDROID MONKEY configuration, it is possible to set the number of interactions (e.g., keystrokes, gestures) to send to the app and the time that must elapse between interaction and the next one.

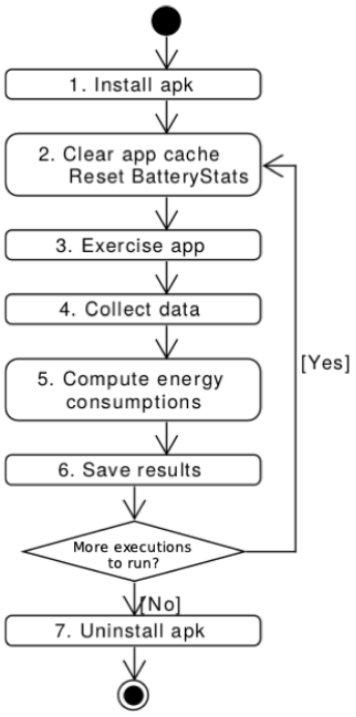

Fig. 2. PETrA work flow

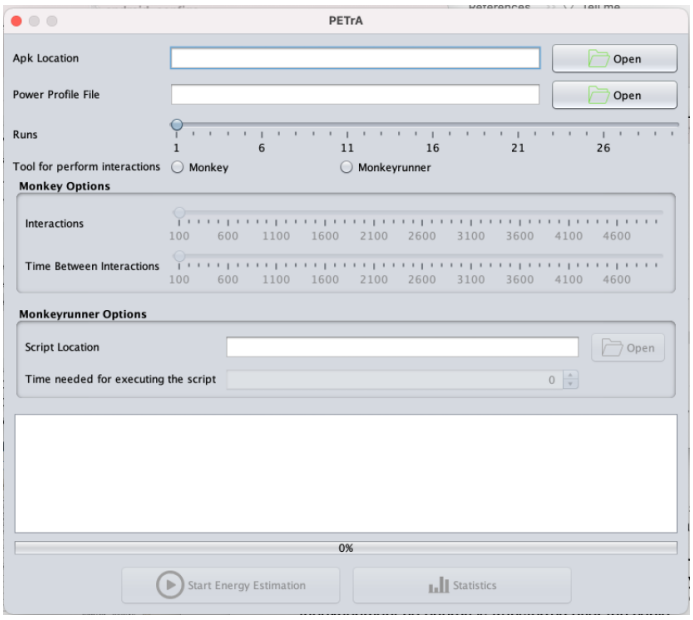

Fig. 3. PETrA tool

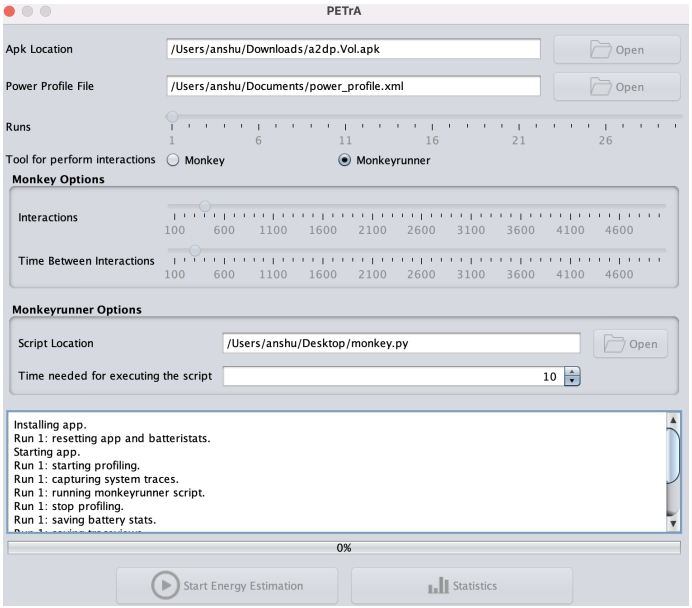

Fig. 4. PETrA tool, installing app and resetting inner process

D. Implementation of PETrA

PETRA's workflow is shown in Figure 2. PETRA starts with a pre-processing phase, which entails installing the app (step 1), cleaning the app cache, and resetting the Android tools that PETrA needs to make its estimations (step 2) [8]. Such pre-processing phases are needed to create a good test environment not influenced by the previous app executions. Subsequently, step 3 of Figure 2 exercises the app using (1) ANDROID MONKEY tool, or (2) an ANDROID MONKEY RUNNER script [?]. ANDROID MONKEY generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as several system-level events to perform stress-testing of the app under analysis. ANDROID MONKEY RUNNER is an API for controlling an Android device [?].

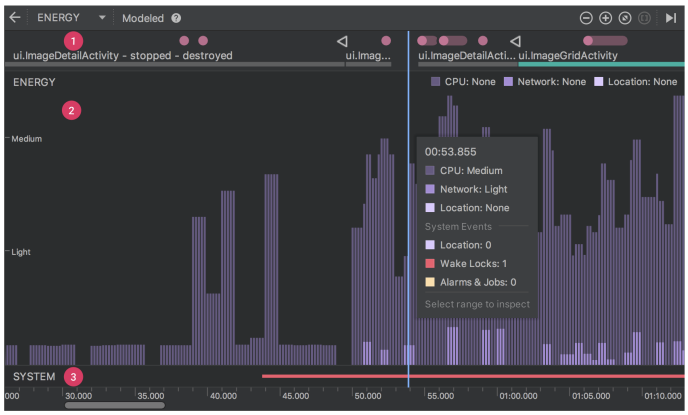

Fig. 5. Android Energy Profiler

E. Implementation of Android Profiler

The Android Profiler in Android Studio 3.0 and higher replaces the Android Monitor tools. The Android Profiler tools provide real-time data to help you to understand how your app uses CPU, memory, network, and battery resources. The Energy Profiler emerge as a row in the Profiler window when you run your app on a connected device or Android Emulator running Android 8.0 (API 26) or higher.

If prompted by the Select Deployment Target dialog, choose the device to which to deploy your app for profiling. If you have connected a device over USB but do not see it listed, ensure that you have enabled USB debugging. Click anywhere in the Energy timeline to open the Energy Profiler. When you open the Energy Profiler, it immediately starts displaying your app's estimated energy usage. To open the Energy Profiler, follow these steps: Select View > Tool Windows > Profiler or click Profile in the toolbar.

We used the Android Profiler and the ten applications, which we tested manually by running each test case five times and taking an average of them. In figure 6, we can see the readings of each application.

V. Inspect network traffic with Network Profiler

When application makes a solicitation to the organization, the smartphone should utilize the power-hungry mobile or WiFi radios to send and get packets. The radios use power to move information or packets. However, it uses additional power to remain alert.

Utilizing the Network Profiler, we can see short spikes of network profiler for spikes, which imply that application requires the radios to turn on as often as possible or to remain alert for extensive stretches to deal with many

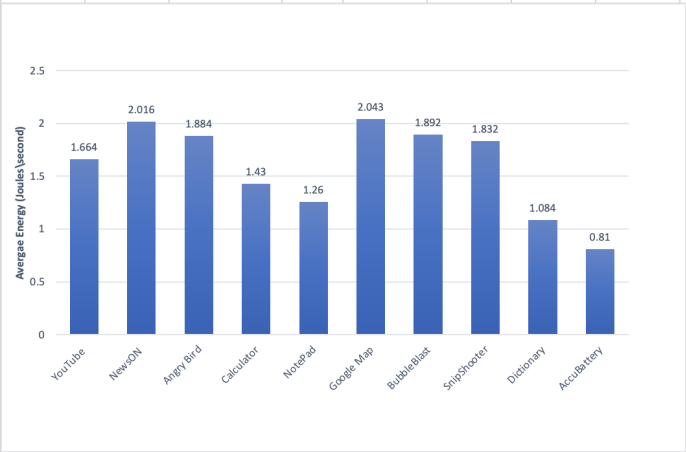

Fig. 6. Average chart of five test run of each app

short demands near one another. This example shows that we might have the option to advance application for improved battery execution by batching network demands, decreasing the occasions the radios should send or get information. This likewise permits the radios to switch into low-power mode to save battery in the more drawn-out holes between grouped solicitations.

A. Network Profiler overview

Android Studio 3.1, as of late, emerged from beta. It has many elements, for example, kotlin build up checks, D8 compiler, and furthermore a redid Network Profiler.

From the beginning of DDMS, we could generally check how network information was being burned-through, yet the current emphasis of profiler has added an entirely different arrangement of provisions. We should look at them.

Most importantly, the network chart looks quite perfect.

B. Energy Estimation while using Network

We can see in figure 8 that when network usage is increasing then energy usage is also increasing.

I have used OkHttp to estimate the network usage sending and receive HTTP-based network requests.

C. OkHttp

OkHttp is a library created by Square for sending and getting HTTP-based network demands. It is based on top of the Okio library, which attempts to be more proficient in perusing data by composing information than the standard Java I/O libraries by making a common memory pool. It is likewise the underlying library for Retrofit library that provides type security for burning-through REST-based APIs.

The OkHttp library executes the HTTP Url Connection interface, which Android 4.4 and later forms currently use. Along these lines, when utilizing the manual methodology portrayed in this part of the aide, the fundamental Http Url Connection class might be utilizing code from the

Fig. 7. Network Profiler

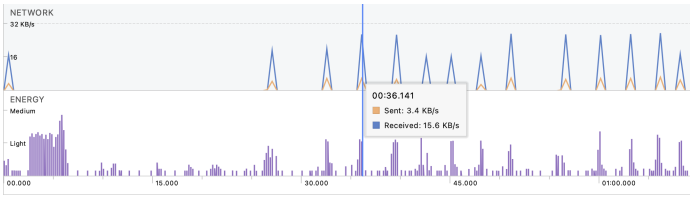

Fig. 8. Energy & Network Profiler

OkHttp library. Notwithstanding, there is a different API given by OkHttp that makes it more straightforward to send and get network demands, which is portrayed in this aide.

What's more, OkHttp v2.4 likewise gives a more refreshed method of overseeing URLs inside. Rather than the java.net.URL, java.net.URI, or android.net.Uri classes, it gives another Http Url class that makes it more straightforward to get an HTTP port to parse URLs and canonicalize URL strings.

VI. Discussion

We presented PETRA and Android Profile software-based tools to estimate the energy consumption of Android apps at method level granularity. The measures of average energy consumption for ten applications from various categories are studied. To understand their energy pattern, we analyzed the most energy-consuming methods in the study's application. Our result shows that the application has a significant variation in the amount of consumed energy.

Some findings blamed built-in systems for retrieving and displaying advertisements with the app for higher energy consumption. Other findings suggested using the .jpeg image format instead of other file types like .gif and

.png. Our findings indicate that poor coding choices in the design of applications are primarily responsible for the higher-than-necessary consumption. Our research is consistent with others. One of the studies revealed that by analyzing and tweaking the design of Wikipedia, energy consumption could be reduced by 30% without affecting the user experience.

Tests were run using an Android smartphone, with Gmail declared as the "greenest" mobile site tested. After researching, it came out that Apple is one of the worst phones in a battery. Nevertheless, this is primarily due to the site not having a version optimized for mobile use.

VII. Conclusions and Future Work

Advancement in power use of smartphone use has become a significant field for research in the present I.T. world. The essential purposes behind battery depleting in cell phones are Network Data Communication like Multimedia Streaming, GPS, WiFi, and Signal Dead Spots. Utilization situations like significant degree of backlight, high-goal Video Playbacks, Graphics, Rich Gaming, and Heavy Computing Processes are the fundamental wellsprings of influence utilization. Different reasons for power wastage are Application Energy Bugs, No-Sleep Bugs, Unnecessary utilization of Sensors, and consistently running Background Processes. Wake Locks and Sensors can likewise rapidly deplete the battery if the software engineers neglect to un-register it on schedule. This examination featured the issues and the answers for the advancement of energy utilization in cell phones.

We presented PETRA and Android Profiler as software-based tools for the estimation of the energy consumption of Android apps at method level granularity. The measures of average energy consumption for ten applications from various categories are studied. Our result shows that the applications have a significant variation in the amount of consumed energy. To understand their energy pattern, we analyzed the most energy-consuming methods in the study's application.

References

[1] Tong Li, Tong Xia, Huandong Wang, Zhen Tu, Sasu Tarkoma, and Pan Hui. Smartphone app usage analysis: Datasets, methods, and applications. IEEE Communications Surveys Tutorials, 03 2022.

[2] Shutong Song, Fadi Wedyan, and Yaser Jararweh. Empirical evaluation of energy consumption for mobile applications. In 2021 12th International Conference on Information and Communication Systems (ICICS), pages 352–357, 2021.

[3] Muhammad Umair Khan, Shanza Abbas, Scott Uk-Jin Lee, and Asad Abbas. Measuring power consumption in mobile devices for energy sustainable app development: A comparative study and challenges. Sustainable Computing: Informatics and Systems, 31:100589, 2021.

[4] Muhammad Umer Farooq, Saif Ur Rehman Khan, and Mirza Omer Beg. Melta: A method level energy estimation technique for android development. In 2019 International Conference on Innovative Computing (ICIC), pages 1–10, 2019.

[5] Mohammad Ashraful Hoque, Matti Siekkinen, Kashif Nizam Khan, Yu Xiao, and Sasu Tarkoma. Modeling, profiling, and debugging the energy consumption of mobile devices. ACM Comput. Surv., 48(3), dec 2015.

[6] Balaji A.Naik and R.K. Chavan. Optimization in power usage of smartphones. International Journal of Computer Applications, 119(18):7–13, 2015.

[7] Abhinav Pathak, Abhilash Jindal, Y. Charlie Hu, and Samuel P. Midkiff. What is keeping my phone awake? characterizing and detecting no-sleep energy bugs in smartphone apps. In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys '12, page 267–280, New York, NY, USA, 2012. Association for Computing Machinery.

[8] Dario Di Nucci, Fabio Palomba, Antonio Prota, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. Petra: A software-based tool for estimating the energy profile of android applications. In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pages 3–6, 2017.