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

Anshu Rathor*, Yaser Jararweh§

Duquesne University, Pittsburgh, Pennsylvania, USA
*rathora@duq.edu, §jararwehy@duq.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. 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.

## II. Background

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 demonstrate burning-through less energy. For instance, an application is supposed to be at higher power use than another application with similar usefulness, yet it can burn through substantially less energy if the runtime is adequately lower than the other application [1].

The paper applies to our work. It analyzes various instruments' profiling abilities and execution and proposes that more work should be done to develop the power further demonstrating the technique. Our work is not the same as theirs as we partition the instruments into equipment and programming-based classes and look at their benefits also, burdens according to the viewpoints of an application's client, engineer, and analyzer [2].

## III. Measuring Power Consumption

Estimating power utilization in smartphones is a difficult task. We need to take uncommon consideration of various equipment parts since the all-out power devoured by the gadget is conveyed among these segments. Observing the parts usage at the software level requires a reverse-engineering device as the source codes of the majority of the applications are not accessible. These devices convert APK files into IR (Intermediate Portrayal) to separate the stream and API used in an application.

Power can be estimated at two degrees of granularity: coarse-grained (at the application level) and fine-grained (at the capacity level or guidance level). Coarse-grained estimations will give the general power utilization of the application to help the client and analyzer of an application. Fine-grained will give actual data to investigate an application at improvement time. [3]

Various apparatuses are accessible, and we give a market outline. These apparatuses give a different type of data about the power utilization of smartphones and are principally isolated into two classes:

### A. Hardware-based Measuring tools

These tools are very accurate to provide the correct result, yet they require specific equipment to calculate the power utilization by hardware components. This equipment is expensive and needs accurate installation also. So, they can provide the correct result. An external device, for example, the Monsoon Power meter [1], can be used to

quantify the current drawn straightforwardly from the cell phone. Another approach to utilizing an external tool is to associate an outside gadget and mount a voltage meter over a resistor associated in series with the cell phone's power supply, permitting the current to be determined from the deliberate voltage change. We can utilize a unique battery or an external power supply to gauge the power utilization. External power sensors are the best quality level for cell phone power examination ineffable from their high exactness and precision. We will talk about some external apparatuses that are utilized in writing to quantify the power utilization of cell phones. These apparatuses are generally precise and fill in as the ground truth-values for other programming-based apparatuses.

### B. Software-based Measuring tools

Programming-based instruments are installed in the device or as a plug-in to check the power difference. This numerical model is fabricated utilizing genuine equipment-based devices to plan the power utilization of various segments of smartphones. These parts can be CPU, memory, GPS, show, and sensor. Software-based components are less expensive and simpler to use than hardware-based instruments. However, these it believed to be less accurate [2]. To determine a feasible numerical model of power utilization, there are two principle techniques. One is to utilize the real hardware measuring instrument to quantify the power utilization of every segment at the point when the gadget is being used. The second is to monitor the power change state by introducing programming in the gadget to figure out which activity causes the adjustment of power utilization and for what segment. We will presently examine some product-based power utilization devices.

## IV. Battery Devouring Components of Smart Phone

In this section, we first describe the components that drain the energy. A study published in 2010 by Aaron Carroll and Gernot Heiser measured various critical points of a smartphone. For example, they measured each component, such as the GPS, under different power modes and usage levels. The power drain of each component varies greatly 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. [4].

Smartphones sensors significantly add to the absolute power usage of the gadget. The most widely recognized today is the Accelerometer, Camera, GPS, Compass, Gyroscope, Gravitation sensors, Light sensor, Proximity sensors, Pressure sensors and so forth Versatile equipment, for example, Network Module, Flash Card, RAM, Processor, Audio and Video Codecs are likewise burns-through parcel of battery power. Figure 1 shows the significant parts of the cell phone that uses battery power. In light of the review via Carroll and Heiser, the power usages of cell phone parts are talked about beneath.

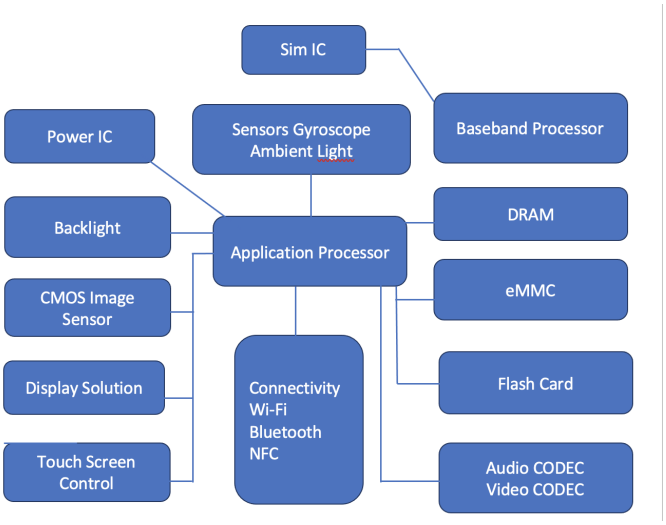


Fig. 1. Power consuming components of Smartphone

### 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

CPU and RAM matter more than probably anything else here because it runs the whole phone. They both use much power in the overall system, approximately around 15-percent, increasing with usage. Its core part needs to be awakened at all times as long as the phone is switched on. Specific processes such as video streaming, internet browsing, CPU utilization rise it starts consuming more battery. RAM & SD Card Both are memory units and use around 7-10-Percent each of the total battery power while performing memory read/write operations

### 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

The power devoured by the GPS module in three circumstances as displayed beneath in Table 1 utilizing just the inward receiving wire, with an outside dynamic radio wire connected, and when inactive.

| 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 utilization, the audio output to a Bluetooth stereo headset was played. The power difference between this and the baseline audio benchmark should yield the utilization of the Bluetooth module. The headset was placed approximately 30 cm from the phone and 10m in the far benchmark, as shown in the following Table.

| 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. USAGE SCENARIOS

1) Audio playback : The 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 : A video document of 5 minutes was recorded without sound and played on a similar stage to quantify the power utilization. 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 call : 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) Taking Photo Cost Energy: 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 DRAINING

There are many aspects to look at the problem of high energy utilization in smartphones. Some of the significant causes of battery-draining are discussed below.

H. 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.

I. Wake lock Registration Bugs

Applications need to acquire a wake lock from OS and specify an awake level to keep a smartphone awake for operations. A full wake lock can keep a CPU awake and its screen at full brightness. The acquired wake lock must be released as soon as the computation finishes execution. Forgetting to un-register wake locks can quickly drain the battery.

J. Sensory Data Under utilization

Sensory data are acquired at the cost of battery power. These data must be effectively used by applications to produce benefits to the phone user. When an application's program logic becomes complex, sensory data may be "underutilized" in specific executions. In such executions, the power cost for acquiring sensory data may outweigh the actual usages of these data.

K. No-Sleep power bugs in applications

No-sleep bugs are power bugs resulting from miss-handling power control APIs in an app, which keeps smartphone battery usage active.

1) 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.[5] 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.[5] 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.

## V. Energy estimation tools used in this research

### A. PETrA

PETrA [6] is software-based solutions are easier to use, but they are possibly less precise than the hardware-based solution. In this demo, we present PETRA, a novel software-based tool for measuring the energy consumption of Android apps. Concerning other tools, PETRA is compatible with all smartphones with Android 5.0 or higher, not requiring any device-specific energy profile. We also provide evidence that our tool can perform similarly to hardware-based solutions.

### B. Android Profiler

The Android Profiler in Android Studio 3.0 and high replaces the Android Monitor tools. The-droid Profiler tools provide real-time data to help you to understand how your app uses CPU, memory, network, and battery resources. The EnergyProfiler appears 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.

### 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.

### D. Implementation of PETrA

PETRA's workflow can be observed in Figure 3. PETRA starts 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 the estimations (step 2).Such pre-processing phases are needed to create a good test environment not influenced by the previous app executions. Subsequently, step 3 of Figure 1 exercises the app using (1) ANDROID MONKEY tool4, 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. It allows writing programs

### 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

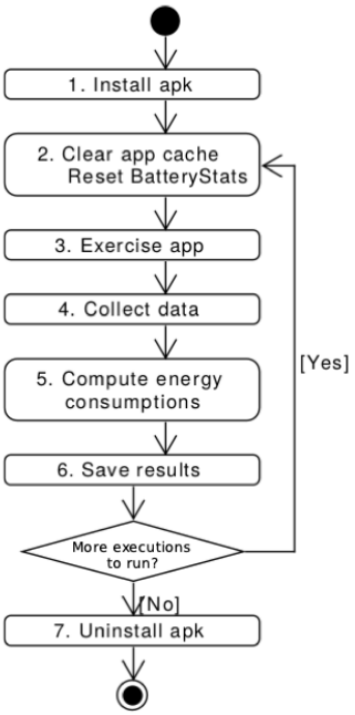


Fig. 2. PETrA work flow

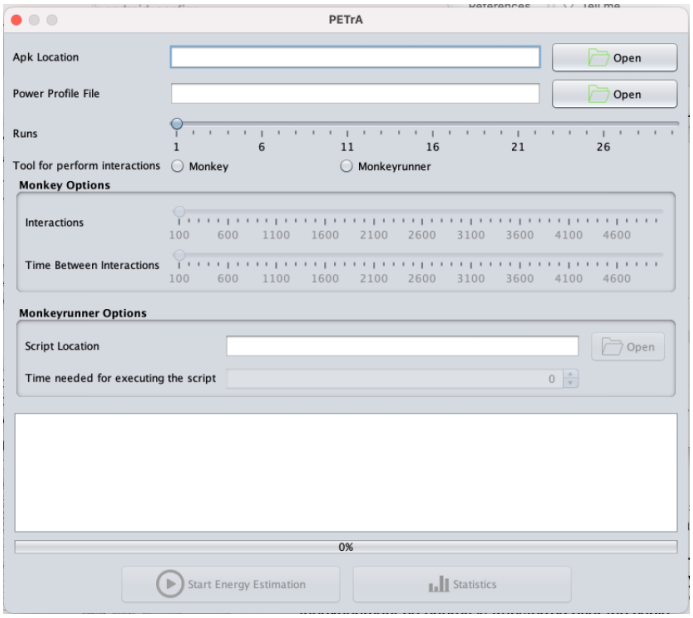


Fig. 3. PETrA tool

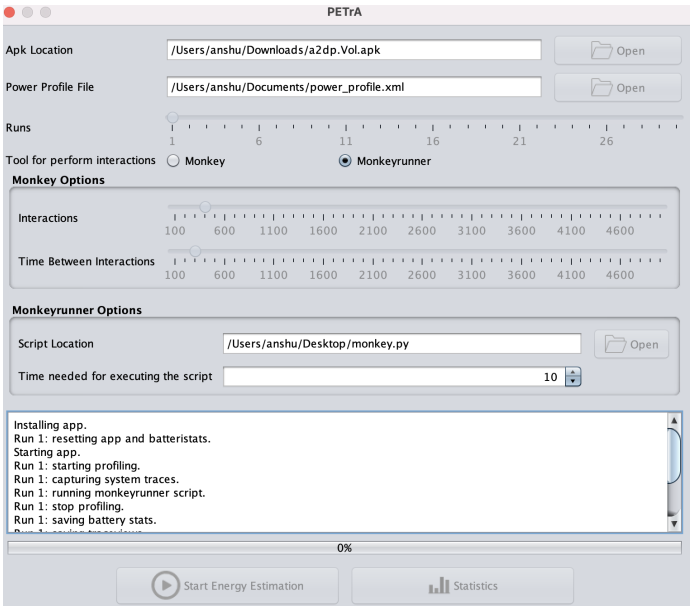


Fig. 4. PETrA tool, installing app and resetting inner process

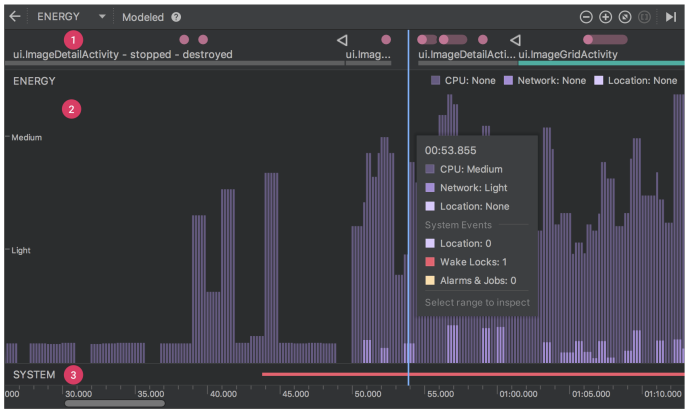


Fig. 5. Android Energy Profiler

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.

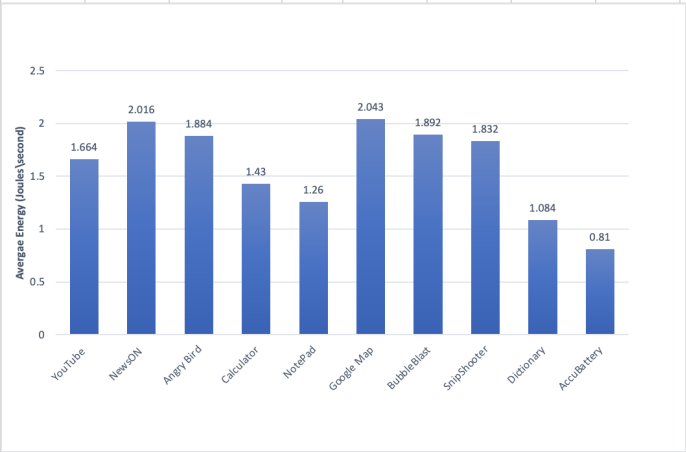


Fig. 6. Average chart of five test run of each app

## VI. Inspect network traffic with Network Profiler

Why you should profile your application's network movement?

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 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.


Fig. 7. Network Profiler

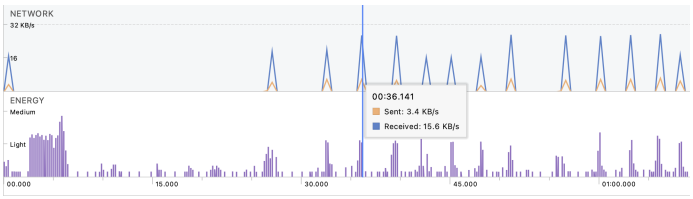


Fig. 8. Energy & Network Profiler

### 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 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.

## VII. 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 Google Mail 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.

## VIII. 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 Profilesoftware-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.

We hope this paper demonstrated the importance of building a mobile application and site optimized for mobile devices.

## References


[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.