CYBERNETICS AND PHYSICS, VOL. 12, NO. 1, 2023, 42–50

FREQUENCY-INDEPENDENT SMARTPHONE PERIPHERALS ENERGY CONSUMPTION ESTIMATION

Ilya Kuznetsov
Department of System Programming
Saint Petersburg State University
St. Petersburg, Russia
kuznetsov.ilya.alexandrovich@gmail.com

Vladislav Miroshnikov
Department of System Programming
Saint Petersburg State University
St. Petersburg, Russia
vladislaw.miroshnikov@gmail.com

Stanislav Sartasov
Department of System Programming
Saint Petersburg State University
St. Petersburg, Russia
Stanislav.Sartasov@spbu.ru

Article history:
Received 11.11.2022, Accepted 25.12.2022

Abstract

The power consumption of mobile devices is a hot topic these days, and it is important to address it when developing applications. One of the most popular ways to measure it is accessing internal sensors using Android Debug Bridge (ADB). We discovered that measurement frequency may skew the power readings. Based on this approach we propose our own algorithm for calculating smartphone energy consumption constants — the power in milliamperes at nominal voltage for different peripherals states. Our algorithm takes measurement frequency bias into account, and its results are compared with the method previously published in literature as well as the baseline data from power profile. We conclude that the developed approach provides better estimation.

Key words

Android, Power Profile, Energy Consumption, Power Constants, Wi-Fi, Bluetooth.

1 Introduction

Our time is unimaginable without mobile devices. Due to their high portability and versatility, smartphones, tablets, and smartwatches are consistently spreading throughout the world. In 2021 there were already more than 6.3 billion smartphone users, and in 2022 this figure is expected to exceed 6.6 billion people [Statista, 2021b]. Today, the most common operating system for mobile devices is Android [Statcounter Global Stats, 2022].

The battery capacity of mobile devices is limited, so the issue of evaluating and optimizing power consumption is relevant. On the hardware side, device manufacturers apply more advanced technologies to create high-capacity batteries and install sensors to monitor power consumption. Software manages the power consumption of device modules based on monitoring using various algorithms. The software also allows obtaining power consumption models and profiles, as well as to perform its refactoring [Sahin et al., 2012].

From a practical standpoint estimating energy consumption might be simpler than directly measuring it because no additional metering hardware is required. Such estimation might be done as a sum of the time various peripherals spent in particular power states multiplied on power constants presented in the device power profile. By convention [Google, 2009] the manufacturers are obliged to provide these values, but in most cases this requirement is ignored. Therefore, the task of determining constants when they are absent is a relevant and important one, and various algorithms and approaches exist to solve it.

In our research we concentrate on determining the power consumption constants and building an energy profile for Wi-Fi and Bluetooth modules as they are among the most battery consuming peripherals. While this task was previously solved by Saksonov [Saksonov, 2014], it was found that Android Debug Bridge (ADB) [Android doc., 2022] — a de-facto standard software tool to get information about the energy used by smartphone peripherals — requires well-thought measurement methodology in order not to skew the power readings. Based on this consideration we present our own

approach to estimate power constants and compare them both with baseline data from power profile and the results obtained from Saksonov's algorithm (SA).

This paper is organized as follows. Section 2 gives an overview of the different methods for obtaining power consumption information, the power profile of the device and approaches to calculate energy consumption constants. Section 3 contains a research on the ADB tool and an experiment showing dependence between power reading frequency and the readings themselves. Section 4 experimentally investigates the found dependence. Section 5 describes our approach to calculate power consumption constants of the smartphone. Its results are compared with SA and power profile constants in Section 6. Section 8 gives general conclusion of the research and further plans.

2 Background

2.1 Methods to obtain energy consumption data

Energy consumption may be either measured or estimated. In both cases one should first obtain raw consumption data and then either calculate the end result or make an estimation model (i.e. determine coefficients for weighted sum). When working with Android OS there are three methods to obtain this raw data: through external measurement device, through Android OS API calls or through ADB.

External measurement device While the exact device might differ from paper to paper, its modus operandi is the same. For example, in [Saksonov, 2014] a Yoctopuce Yocto-Amp USB Electrical Sensor [Yoctopuce, 2022] ammeter was used to measure the power consumption of the device modules, which can be programmed to perform a specific test scenario. Other ammeters or more functional multimeters can also be used such as Monsoon power monitor [Monsoon Solutions, Inc., 2022]. External measuring devices have a considerable disadvantage: obtaining energy consumption data requires direct physical intervention with the device. Many modern devices do not allow access to the battery without special preparations.

Constructing a device energy consumption model using Android OS API calls As in the PowerTutor API [PT, 2011], information on device modules power consumption in different states can be obtained from the corresponding API calls. Then, by correlating the measured power consumption with the pre-defined data, namely peripherals hardware power states, a power consumption model of the device is constructed [Zhang et al., 2010]. For example, for the Wi-Fi module pre-defined power states includes uplink channel rate, uplink data rate and packets transmitted per second. The disadvantages of this approach are that under Android 7.0 (Android API Level 24) and higher this API requires superuser access rights, and many system calls are marked as "Deprecated".

In the article [Hu et al., 2018] is also proposed an approach with construction of the Android device energy consumption model. The authors developed an algorithm that allows to perform a "lightweight" analysis and prediction of the Android application energy consumption. This method does not require the source code of the application, but allows users to perform tooling and implement in the APK file a special "monitoring" calls that provide an ability to log method calls and API of the application. Next, based on linear regression analysis and using ADB calls, the energy consumption is predicted at the Method Level and API Level of the application until the desired accuracy is achieved using three statistical metrics: the multiple correlation coefficient, the average value of relative error, and the standard deviation of relative error.

Android Debug Bridge ADB is a command line tool that allows interaction with the smartphone and provides access to UNIX shell for running various commands on the device. The ADB API is supported and receives updates in the most recent versions of Android OS. However, its power readings are as precise as on-board power sensors.

In the end, ADB is a good enough alternative for estimating power consumption constants when an external device is not available.

The command `adb shell dumpsys batterystats` outputs information about the power consumption of the modules as well as their consumption in the context of each process [Android doc., 2021]. This includes general information about how much power the respective module has consumed, for how long it has been running, and some other data with a report from the RESET point, which is responsible for resetting the device's battery consumption statistics. It is automatically set when fully charged, but can also be set manually with the `adb shell command dumpsys batterystats --reset`.

Attention should be paid to the `Statistics since last charge` block. It contains information about the energy consumed during Wi-Fi and Bluetooth operation. The data is stored in the fields `Wi-Fi Battery drain` and `Bluetooth Battery drain` respectively.

For example, on a Samsung Galaxy S9+ (Android API Level 28) test device, the following result was obtained after outputting the information with the command.

```
WiFi Battery drain :  12.363808 mAh
Bluetooth Battery drain:  1.86 mAh
```

After resetting the statistics the field values are set to zero or not displayed at all until actual consumption is shown. Below are the values after using `adb shell dumpsys batterystats --reset` command with Wi-Fi and Bluetooth enabled on the same test device.

```
WiFi Battery drain :  0.101896 mAh
```

`Bluetooth Battery drain: 0.0506 mAh`

The next important output block is `Estimated power use (mAh)`. It shows a wide range of power consumption values, including information for the various peripherals of the device, as well as information about each individual process, exactly how much the CPU, Sensor, and many other modules, including Wi-Fi and Bluetooth, are consuming in the context of its operation. In addition, Wi-Fi and Bluetooth are presented here as processes, and it's possible to see detailed information for them, which extends the `Statistics since last charge` block. All data here is also counted from the RESET point.

After RESET, with Wi-Fi and Bluetooth enabled, the output on a test device is as follows.

`Wifi: 0.00228 ( cpu=0.000645 wifi=0.00164 ),`

`Bluetooth: 0.001947 ( cpu=0.000967 bt=0.00098 ).`

In the article [Armendáriz Irigaray, 2019] was applied the ADB tool approach. In this paper is considered the creation of an application that allows users to evaluate the power consumption of different modules in certain test scenarios. For example, to test the Wi-Fi module was implemented a test scenario with a 500 MB file transfer via Dropbox. For each of the test scenarios, 30 runs are performed and, using the ADB API calls, the average power consumption value is collected and displayed.

2.2 Energy profiles of devices

Power Profile (`power_profile.xml`) is an XML file provided by the manufacturers in a specific directory containing constant values for various peripherals' energy consumption in a specific Android device. If a particular manufacturer doesn't provide those values, the content of this file is initialized with default values from Google [Google, 2009]. The profile specifies the power in milliamperes (mA) at nominal voltage.

The main focus of this article are Wi-Fi and Bluetooth modules in Android devices as they are actively used in everyday life [Statista, 2021a] [Bluetooth, 2021]. According to the power profile structure, the following energy constants are defined which correspond to different peripherals states:

wifi.on Power consumed when the Wi-Fi module is on but not transmitting or receiving data. According to Google's recommendations, this value can be measured by the difference between the power consumption when the system is suspended with Wi-Fi turned on and turned off.

wifi.scan Power consumed during Wi-Fi access point scanning.

wifi.active Power used when transmitting or receiving data over Wi-Fi.

bluetooth.on Power when the Bluetooth module is switched on and in search mode for available devices, but not connected to any device.

bluetooth.active Power when receiving or transmitting data via Bluetooth.

2.3 Approaches to determine energy consumption constants

Several other algorithms are described in the literature to determine power constants for specific smartphone modules. A power model was shown to provide good estimations for Wi-Fi and GPS modules [Bareth, 2012]. CPU power consumption can be estimated with non-linear model [Myasnikov et al., 2021]. However, a direct predecessor for this research is SA [Saksonov, 2014], and in order to determine the energy consumption constants using this method, the following steps must be followed:

1. Measure the data at the selected sampling rate using external measuring device or ADB.
2. Calculate the mean a and the standard deviation s from the data.
3. All values that are not within the interval $[a-s; a+s]$ are no longer considered.
4. Calculate the expectation of the points c that are in part of the segment $[a-s, a]$. The resulting value is the power consumption in mAh, then it is converted to mA to match the standard units for the power profile.

3 Android Debug Bridge and Sampling Rate Influence

3.1 Reasoning behind experimentation

The phenomenon under discussion was first encountered when building an energy consumption estimation model for Wi-Fi and Bluetooth peripherals for a broader research. Initially, SA was chosen to evaluate energy consumption constants for each module state, and the following steps were taken to ensure the test scenario power metering will provide proper values.

In the end, different constant values were obtained in different experiments, and after ruling out the possibility of mathematical or algorithmic error we've found that the notable methodological difference between original SA and our scripts was ADB sampling rate. Thus, we decided to investigate the power consumption of Wi-Fi and Bluetooth modules in different operating modes with `adb shell dumpsys batterystats` command over a fixed time interval at different sampling rates.

3.2 Experimental methodology

A test scenario was created to test each specific module energy state for 60 seconds, and multiple test runs were conducted for a specific set of sampling rates. The following list was selected based on preliminary observations (values are presented in Hz from highest to lowest): 4, 2, 1.33, 1, 0.66, 0.5, 0.4, 0.33, 0.28, 0.25, 0.22, 0.2, 0.16, 0.14, 0.12, 0.11, 0.1. The number of measurements was calculated as frequency multiplied by 60.

To reduce the noise from background processes and other peripherals the following steps were taken:

1. Put all applications, except those directly involved in the experiment, into (deep) sleep mode, prohibiting any background activity and sending notifications.
2. Disabling mobile phone communication, activating air mode.
3. Disabling device charging using `adb shell dumpsys battery unplug` command.
4. Disconnect device screen using `adb shell input keyevent command 26`.
5. Enabling only the module under test to prevent components from affecting each other. This action is performed with the `adb shell svc [module] enable` command.

We consider those steps to be in line with other works in the field, and it was experimentally verified that on the two tested devices Samsung Galaxy S5 (Android API Level 23) and Samsung Galaxy S9+ (Android API Level 28) a "clean" zero is displayed in the relevant fields when no power consumption by the modules is present, or the fields equivalently remain hidden.

An appropriate `bash` script was written for each test case. It performs isolation steps and then collects measurement results at time points corresponding to sampling rate. Power consumption values are extracted from `Wi-Fi Battery drain` and `Bluetooth Battery drain` fields for the module under examination.

For each sampling rate appropriate regression lines expressing the best prediction of the power consumption dependent variable on the time point independent variable were constructed, as well as the confidence intervals showing the estimate with a given confidence interval of 95%.

All test runs were performed on a Samsung Galaxy S9+ device (Android API Level 28).

3.3 Experimental results

Testing the Wi-Fi module in wifi.active mode The Wi-Fi module was switched on, the device was connected to the network, and then the YouTube broadcast was played at 1080p resolution and the screen was switched on, which required the video to be played back. There was no additional delay before testing, as the Wi-Fi scanning takes place before connecting to the network and therefore could not be affected in this experiment.

The following graphs were plotted using regression analysis and confidence intervals for the sampling rates from 0.1 to 4 Hz. For other frequencies similar construction was performed.

The resulting regression lines show that energy consumption over time may be approximated as a linear function, but the slope is different for each sampling rate.

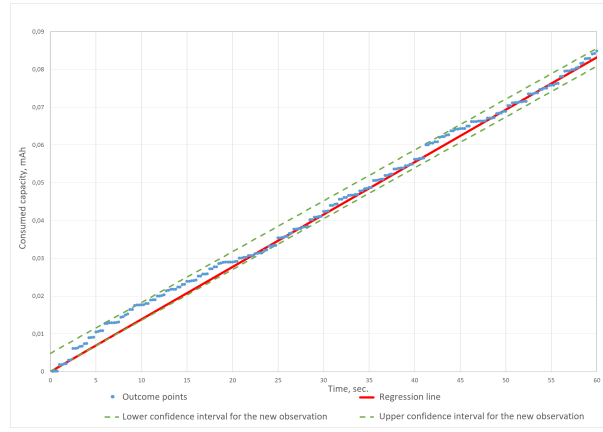

Figure 1. Information on Wi-Fi module power consumption in `wifi.active` at a sampling rate of 4 Hz

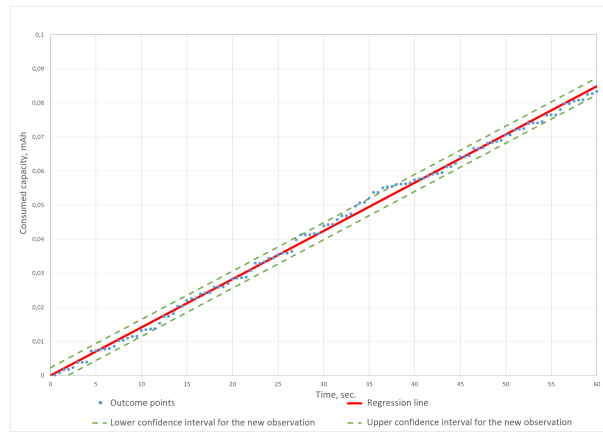

Figure 2. Information on Wi-Fi module power consumption in `wifi.active` mode at a sampling rate of 2 Hz

Testing the Wi-Fi module in wifi.on mode The device had Wi-Fi on, was not connected to the network and, unlike the previous experiment, had a 30-second time interval to skip the scan, which starts automatically when Wi-Fi is switched on.

Testing the Wi-Fi module in wifi.scan mode This mode has not been tested as there is no command to scan Wi-Fi networks for the specified Android API level.

Testing the Bluetooth module in bluetooth.on mode The device had Bluetooth on and was not connected to the points, with a time interval of 30 seconds to skip the scan, which starts automatically when Bluetooth is switched on.

Testing the Bluetooth module in bluetooth.active mode The Bluetooth module on the device was switched on and it was connected to a wireless portable audio system in playback mode of the pre-loaded audio. A time interval of 30 seconds is unnecessary as the Bluetooth points are scanned before they are connected and therefore could not be reflected in this experiment.

We omit here graphs for `wifi.on`, `bluetooth.on` and `bluetooth.active` test cases for the sake of brevity, but a trend similar to `wifi.active` experiment was observed: the higher sampling rate is, the higher recorded power consumption.

3.4 Conclusions for experiments

First and foremost, our experiments show that energy consumption over time may be reliably approximated by a linear function for each of the peripherals' states we were able to conduct experiments for. While real-world scenarios seldom use Wi-Fi or Bluetooth for data transfer for the entire scenario duration, slope coefficient is a good upper bound estimate.

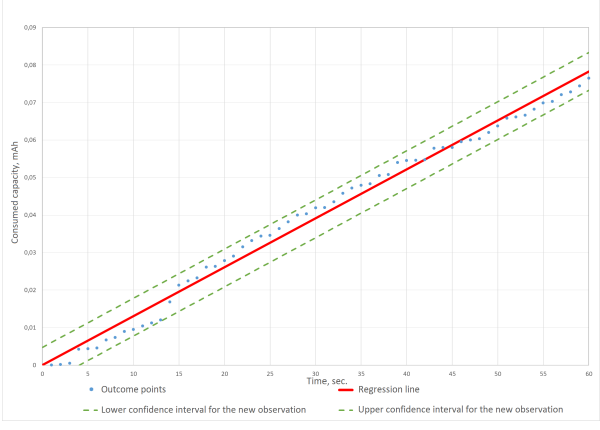

Figure 3. Information on Wi-Fi module power consumption in `wifi.active` mode at a sampling rate of 1 Hz

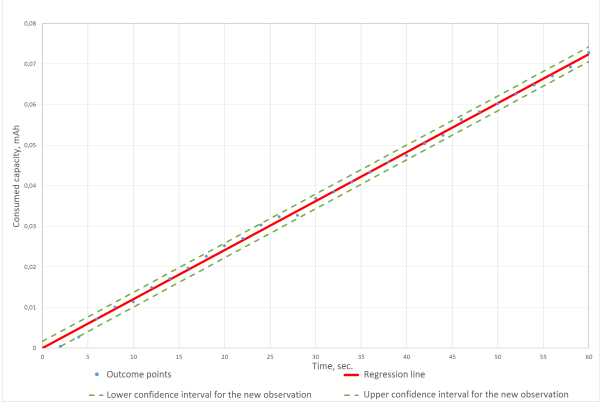

Figure 4. Information on Wi-Fi module power consumption in `wifi.active` mode at a sampling rate of 0.5 Hz

Second, qualitative analysis of our results is straightforward and in a sense self-evident: increased sampling rate introduces more overhead from sampling tool itself — ADB in our case. However, our experiments also allow us to investigate this relationship in quantitative way — plotting of regression lines slope coefficients as a function of frequency will be indicative.

4 Relation between ADB overhead and sampling rate

To conduct a quantitative analysis, another set of `bash` scripts was written as an extension of previous ones and the same preparation steps as described in Section 3.2 were done. This time, however, the experiment was run on two test devices: Samsung Galaxy S21 Ultra (Android API Level 30) and Sony ZL (Android API Level 22). We assumed that difference in particular peripherals model and Android OS versions could be indicative whether relation between sampling rate and ADB overhead is the same or similar among different smartphones or not.

For each of the states `wifi.on`, `wifi.active`, `bluetooth.on`, `bluetooth.active` testing was conducted at the frequencies shown in the Table 1 below. Experiments in `wifi.scan` mode were not conducted due to the absence of commands for the specified Android API levels.

To improve the accuracy of the experiment, 5 test runs were performed for each frequency, and the slope coefficient for each regression line was calculated. Then for each frequency the mean and median of the slope were calculated. We chose two modes of calculation because 5 values obtained for each frequency allow to average its slope coefficient in order to minimize the external effects in different ways in addition to the experiment's isolation.

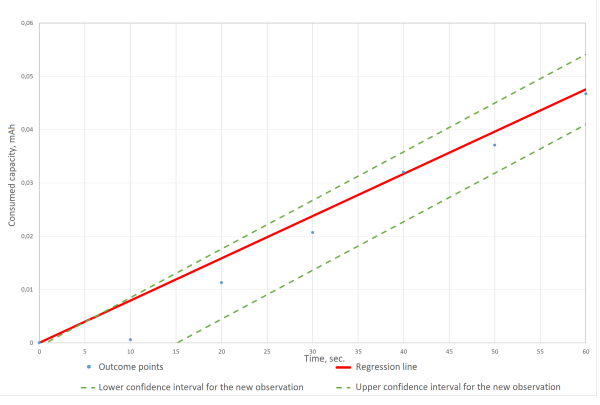

Figure 5. Information on Wi-Fi module power consumption in `wifi.active` mode at a sampling rate of 0.1 Hz

Table 1. Sampling rates and number of measurements per 30 seconds for one test run

Sampling rate (Hz)	Number of measurements (pcs.)
10	300
5	150
3.33	100
2.5	75
2	60
1.67	50
1.43	42
1.25	37
1.11	33
1	30
0.91	27
0.83	25
0.77	23
0.71	21
0.67	20
0.625	18
0.59	17
0.56	16
0.52	15
0.5	15

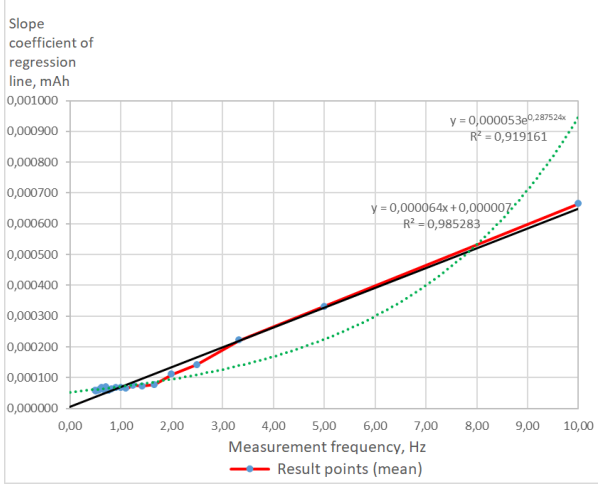

Figure 6. Relation between the slope of the power consumption regression line and measurement frequency in `wifi.on` mode (SG S21 mean)

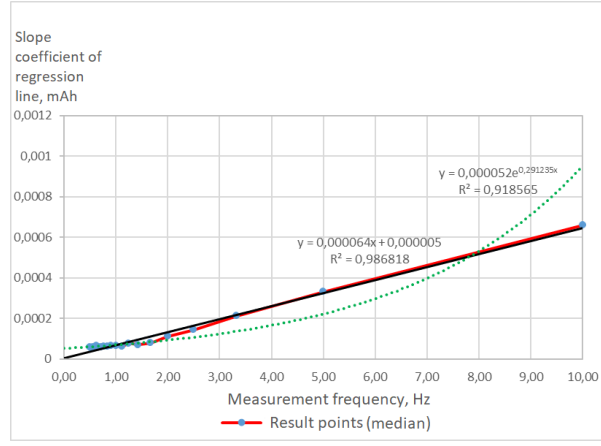

Figure 7. Relation between the slope of the power consumption regression line and measurement frequency in `wifi.on` mode (SG S21 median)

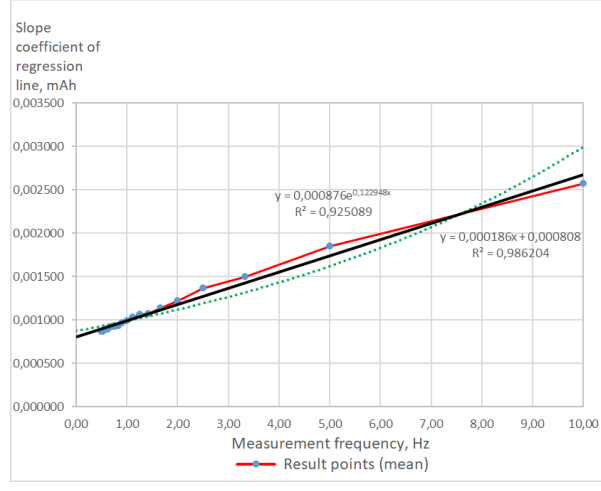

Figure 8. Relation between the slope of the power consumption regression line and measurement frequency in `wifi.on` mode (Sony ZL mean)

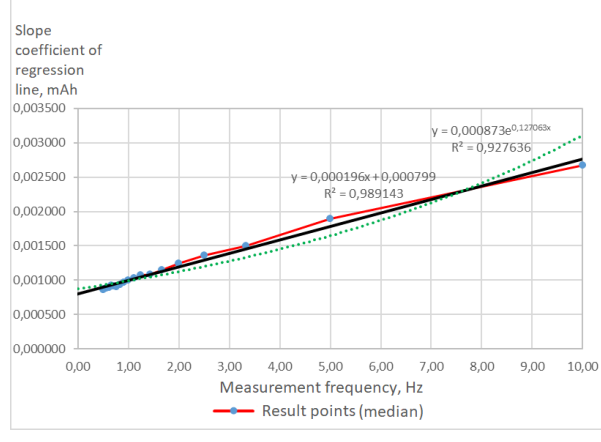

Figure 9. Relation between the slope of the power consumption regression line and measurement frequency in `wifi.on` mode (Sony ZL median)

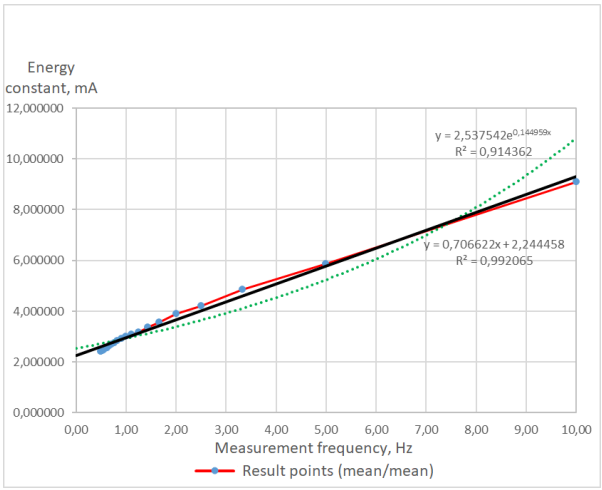

Figure 10. Relation between power consumption constant and sampling rate for `wifi.on` state (mean/mean)

The results were plotted using linear and exponential regression analysis. We chose exponential function over power functions because the researched frequency interval is compact, and exponential regression could produce similar results to power regression while at the same time being more generic. These were extrapolated at the point of intersection with the ordinate axis based on the experimental results from Section 3.3.

Again, for the sake of brevity, we are showing only the experiments for `wifi.on` state. Other test scenarios such as `wifi.active`, `bluetooth.on` and `bluetooth.active` follow a similar trend. Results for Samsung Galaxy S21 Ultra (Android API Level 30) are drawn in Fig.6-7, and results for Sony ZL (Android API Level 22) are drawn in Fig.8-9.

We take a special note on points where regression lines intersect Y-axis. While these points cannot be obtained physically as they correspond to a sampling rate of 0 Hz, that is, an estimation where no samples were taken at all, they are nonetheless important as they show a hypothetical peripheral power consumption without using ADB. Therefore, by constructing regression lines and evaluating which one better approximates obtained slope values, we could also extrapolate it to obtain "clean" peripheral power consumption in a particular state.

Starting at 2 Hz linear regression gives the closest approximation compared to exponential regression on both devices, which approximates better at low frequencies. This is also confirmed by the determination coefficients shown in the graphs. As linear regression demonstrates the maximum likelihood ratio, in this set of experiments we choose it to determine the power constant for a peripheral state.

While the presented graphs tend to be better approximated with linear regression, in some cases extrapolation from exponential regression might provide better results. In Fig.6 linear regression intersects the Y-axis close to zero which is not feasible from a physical standpoint, and exponential regression provides better estimation in this case. Anecdotal evidence suggests that for older smartphone models linear increase in sampling rate results in exponential increase in power consumption, making linear regression not applicable.

Overall, as a rule of thumb to reduce the noise from higher frequencies we suggest constructing regression lines based on points from 0.5 to 2 Hz.

5 Frequency-independent algorithm for calculating energy constants

5.1 Algorithm description

Based on the previous sections we propose the following algorithm to take into account energy measurement bias occurring at a particular measurement frequency, estimate energy constants and construct the power profile of the device if it is not available. Note that while this algorithm as initially developed for Wi-Fi and Bluetooth modules we consider it to be general enough to be used for other peripherals as well. Each peripheral state energy constant should be determined by a separate run of an algorithm.

1. Create a test case for the selected peripheral energy state (i.e. continuous data transfer via Bluetooth) which records energy consumption of a peripheral at a selected sampling rate.
2. Run test case n times for each of the sampling frequencies specified in Table 1.
3. From all recorded energy consumption readings calculate power rate by dividing consumed energy to its timestamp from the beginning of the test run. Find mean or median power rate for the test run.
4. Find mean or median power rate among test runs for a selected sampling rate.
5. Construct a regression line (linear or exponential) for power rates at sampling rates and extrapolate "clean" power rate value at 0 Hz.

Our experiments show that selecting the same averaging approach at both steps 3 and 4 ("mean/mean" or "median/median") produces the most consistent results comparing with SA approach and taking into account power profile of the device, which is also confirmed by the maximum likelihood ratio obtained in the regression analysis.

5.2 Example of calculation

As an example, we've written a `bash` script for the described algorithm which also includes the aforementioned isolation steps and assume that n is 5. The experiment took place on a Sony ZL device (Android API Level 22), and it has a power profile initialized by the manufacturer.

The resulting graphs in Fig.10 and Fig.11 show the power consumption constants obtained by the proposed algorithm. The coefficient of determination shows that the linear regression gives a better approximation than

Table 2. Table of energy consumption constants

Test algorithm / data source	wifi.on (mA)	wifi.active (mA)	BT. active (mA)
SA 5 Hz	1.15	0.8	9.9
SA 1 Hz	0.34	0.17	2.97
Mean / mean / linear	2.24	2.2	28.13
Mean / mean / exponential	2.54	2.35	41.7
Median / median / linear	2.46	2.41	30.1
Median / median / exponential	2.75	2.56	35.8
Power profile	2.7	26	27.7

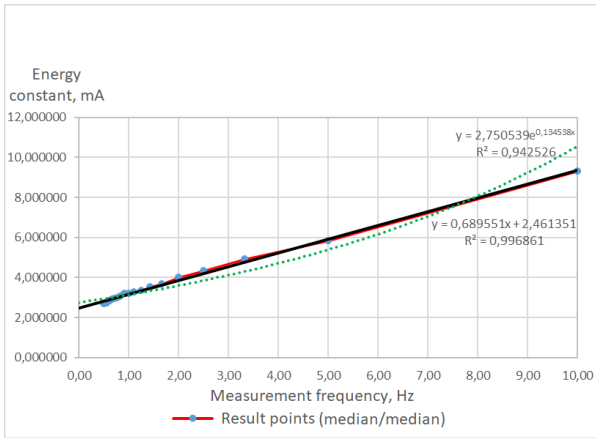

Figure 11. Relation between power consumption constant and sampling rate for `wifi.on` state (median/median)

the exponential one. Note that the values obtained from mean/mean and median/median approaches do not differ significantly.

For other test scenarios, the energy consumption constants have been determined in a similar way. The exception is the Bluetooth module in `bluetooth.on` mode, as on the device under test the ADB always gave a zero power consumption value. Note that mentioned ADB commands provide only the data for Wi-Fi and Bluetooth components, so, for example, CPU power readings are deliberately excluded, as their contribution to total energy consumption might be considerable [Bogdanov et al., 2021].

6 Algorithm comparison

As stated before, we consider our work to be a continuation of SA approach, therefore we compare results of our approach to SA results as well as the constants provided by manufacturer in `power_profile.xml`.

Testing was done on the Sony ZL device (Android API Level 22) which has an extensive power profile from the manufacturer. Values for our method were obtained from the calculation described in Section 5.2. SA algorithm was implemented in a `bash` script. Separate sets of SA test runs were obtained under sampling rates of 1 Hz and 5 Hz. The results for both methods were converted from mAh to mA and compared to power profile constants. Results are shown in Table 2.

Our results demonstrate that SA at 5 Hz produces higher constants than at 1 Hz. For `bluetooth.on` mode it is not possible to obtain a constant because the device displays zero power consumption. It is not clear whether it is due to API limitations or due to the device optimizing the battery consumption. The values of the constants obtained by our algorithm are significantly closer to the values specified in the actual power profile of the test device than those calculated by SA except `wifi.active`. It is possible that the value in power profile is wrong, but there is also a possibility that our test case doesn't load Wi-Fi module to maximum. Video and audio streaming is usually done in data transfer bursts as this approach saves more energy than consistent Wi-Fi usage. If it is true, then careful design of a test case is of utmost importance. It can also be assumed that `wifi.active` test scenario may be affected by the fact that the amperage values may differ from the rated amperage, and electronic components always have performance tolerances. Nevertheless, based on the collected data we consider our algorithm to provide better estimation of power constants than SA approach.

7 Applicability of the developed approach

Our approach has been implemented in the open-source energy profiler [Navitas Framework, 2022] and can be applied to estimate energy consumption and derive energy constants on the user's Android smartphone using pre-defined or custom test scenarios. This allows users to implement a suitable test case for their purposes in order to achieve the most accurate estimates of energy consumption or constants obtained with our method.

8 Conclusion and future work

The scripts referenced in this work are openly available[1][2].

To summarise our contribution, while it was known that ADB imposes overhead when using it for energy

[1] `https://docs.google.com/spreadsheets/d/1CWjhzBHlUclDs2EKD_uKymNy5BPSjoQNTvve97tBK5E/edit?usp=sharing`

[2] `https://github.com/Stanislav-Sartasov/Navitas-Framework/tree/wifi_bluetooth_research`

consumption investigations, this overhead wasn't estimated for a given sampling rate and results were not adjusted. Simply changing sampling rate would introduce result skewing in one way or another, therefore it is necessary to compile a list of frequencies, obtain data for each of them, and then apply the best approximation depending on the given module and target device. Our algorithm is built on this observation and provides power constants using ADB that are closer to power profile values.

In the future we plan to expand our approach to other device peripherals, e.g. GPS module. However, even with Wi-Fi and Bluetooth support our algorithm is an enabling technology for a broader research in energy antipatterns occurring in mobile software.

Acknowledgements

This work was supported in part by the St. Petersburg State University (project ID 94062114).

References

Android doc. (2021). ADB gathering data with Batterystats. https://developer.android.com/topic/performance/power/setup-battery-historian#gather-data. [Online; accessed 02-February-2022].

Android doc. (2022). Android Debug Bridge (adb). https://developer.android.com/studio/command-line/adb. [Online; accessed 02-February-2022].

Armendáriz Irigaray, T. (2019). Development of a tool to measure smarthphone´s battery consumption.

Bareth, U. (2012). Simulating power consumption of location tracking algorithms to improve energy-efficiency of smartphones. In *2012 IEEE 36th Annual Computer Software and Applications Conference*, pp. 613–622.

Bluetooth (2021). Total Annual Bluetooth Device Shipments in 2021. https://www.bluetooth.com/wp-content/uploads/2021/01/2021-Bluetooth_Market_Update.pdf. [Online; accessed 02-February-2022].

Bogdanov, E., Bozhnyuk, A., Bykov, D., Sartasov, S., Sergeenko, A., and Granichin, O. (2021). Dynamic voltage-frequency optimization using simultaneous perturbation stochastic approximation. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 3774–3779.

Google (2009). Power Profile of Android device. https://android.googlesource.com/platform/frameworks/base/+/master/core/res/res/xml/power_profile.xml. [Online; accessed 02-February-2022].

Hu, Y., Yan, J., Yan, D., Lu, Q., and Yan, J. (2018). Lightweight energy consumption analysis and prediction for android applications. *Science of Computer Programming*, **162**, pp. 132–147. Special Issue on TASE 2016.

Monsoon Solutions, Inc. (2022). High voltage power monitor. https://www.msoon.com/online-store. [Online; accessed 8-February-2022].

Myasnikov, V., Shaposhnikov, A., Sartasov, S., Gordienko, E., Aphonina, O., and Gamaonov, A. (2021). Navitas framework: A novel tool for android applications energy profiling. In *Proceedings of the Sixth Conference on Software Engineering and Information Management 2021 (SEIM 2021)*, CEUR Workshop Proceedings, pp. 1–9.

Navitas Framework (2022). An open-source power profiling framework for Android. https://github.com/Stanislav-Sartasov/Navitas-Framework. [Online; accessed 24-December-2022].

PT (2011). PowerTutor - A Power Monitor for Android-Based Mobile Platforms. http://ziyang.eecs.umich.edu/projects/powertutor/index.html. [Online; accessed 02-February-2022].

Sahin, C., Cayci, F., Manotas, I., Clause, J., Kiamilev, F., Pollock, L., and Winbladh, K. (2012). Initial explorations on design pattern energy usage. *2012 1st International Workshop on Green and Sustainable Software, GREENS 2012 - Proceedings*.

Saksonov, A. (2014). Method to derive energy profiles for android platform.

Statcounter Global Stats (2022). Mobile Operating System Market Worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide. [Online; accessed 02-February-2022].

Statista (2021a). Mobile internet usage worldwide - statistics. https://www.statista.com/topics/779/mobile-internet/#dossierKeyfigures. [Online; accessed 02-February-2022].

Statista (2021b). Number of smartphone users from 2016 to 2021. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. [Online; accessed 02-February-2022].

Yoctopuce (2022). Yocto-Amp ammeter. http://www.yoctopuce.com/EN/products/usb-electrical-sensors/yocto-amp. [Online; accessed 02-February-2022].

Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., and Yang, L. (2010). Accurate online power estimation and automatic battery behavior based power model generation for smartphones. CODES/ISSS '10, New York, NY, USA, Association for Computing Machinery, p. 105–114.