Energy-Efficiency Comparison of Mobile Platforms and Applications: A Quantitative Approach

Grace Metri
Wayne State University
gmetri@wayne.edu

Weisong Shi
Wayne State University
weisong@wayne.edu

Monica Brockmeyer
Wayne State University
mbrockmeyer@wayne.edu

ABSTRACT

Given the number of platforms and apps with similar functionalities, this paper describes the challenges and identifies the gaps toward comparing mobile platforms and apps for energy efficiency. In addition, based on case studies that focus on energy efficiency comparison of different app categories on the most popular platforms, the paper discusses insights related to the major platform providers, energy-efficient app design, and app developers common practices.

1. INTRODUCTION

Today's mobile users face choices of platforms and apps with similar functionalities. Therefore, it is important to understand their relative energy efficiencies. However, comparing platforms and apps from an energy efficiency perspective is a challenging task given the lack of appropriate tools and technologies, possible measurement errors, and designing sound case studies. Despite the challenges, we collected power related metrics on different mobile platforms in order to achieve the following: 1) to quantify the energy efficiency gain of native apps vs. their web counterparts. 2) to quantify the difference in energy efficiency of same app categories on different platforms. The contributions of this paper are:

- We discussed the challenges and identified gaps toward comparing the energy efficiency of platforms and apps.
- Using case studies which focused on energy efficiency comparison of different app categories on the most popular platforms, we derived a list of insights related to the major platform providers, energy-efficient app design, and common practices of app developers.

The paper is organized as follows. We first discuss the challenges toward comparing platforms and apps for energy efficiency in Section 2. We present our experimental approach in Section 3 followed by detailed case studies in Section 4 and a list of insights in Section 5. Next, we present some related work in Section 6 and conclude in Section 7.

HotMobile '15 February 12 – 13 2015, Santa Fe, NM, USA

http://dx.doi.org/10.1145/2699343.2699358.

2. CHALLENGES

Given the large number of choices available for users in terms of platforms and apps, then comparing and ranking the energy efficiency of both can create some sort of competitive advantage. However, such a comparison is not a straight forward task. It is very challenging at best due but not limited to the following:

- Purely comparing the energy efficiency of different platforms is a hard task due to the fact that each platform requires different tools each of which can have distinct capabilities, different accuracy rates, and introduces varying energy consumption overhead. Given the tools' limitation, we do not directly compare the energy efficiency of platforms. Instead, we select a set of apps for each platform and then compare their relative energy efficiency.

- The test environment, if not kept constant, may impact the accuracy of the data. For instance, some devices utilize ambient light sensors that adjust the display backlight based on the surrounding light. As a result, during our data collection, we strived at keeping the lighting consistent across all scenarios. Another factor which can impact the accuracy is the Wi-Fi signal strength. Therefore, we made an effort to perform all tests at consistent Wi-Fi signal strength.

- Eliminating activities of background apps and services was challenging as well. For instance, even after killing Spotify from *Task Manager* on Android, we still periodically observed some related running processes. As a result, we uninstalled from the platform under test all apps that were not installed by default. Despite all our efforts, there were still some services that we were not able to stop. For instance, on Android, we failed at stopping the activities of *Search Application Provider* and *Google Account Manager*. Similarly, on Windows, we were not able to stop many system processes. On the other hand, on iOS, we were unable to identify background processes, not because they did not exist, but because we didn't have the appropriate tools.

- The energy efficiency of an app with dynamic content may significantly differ from one experiment to another. For example, since the energy consumption of *Facebook* depends on the number of status updates of *friends*, then login into an *uncontrolled* account may introduce measurement errors based on inconsistent activities during the experiment. As a result, for our case studies, we created an account which was *controlled* and used solely for the experiment purpose where friends posted identical updates during the experiment.

- Many apps have logic in the cloud which may impact their power consumption. Since our testing model cannot enable us to compare platforms and apps fairly if they use

Table 1: List of devices

Device	OS	Processor	Memory	Storage
Nexus 7	Android 4.3	Qualcomm Snapdragon S4	1 GB	16 GB
iPad Air	iOS 7.0.6	A7 chip with 64-bit	1 GB	16 GB
Surface 2 Pro	Windows 8.1	Intel(R) i5-4200U	4 GB	64 GB

the cloud, we refrained from comparing such scenarios. For example, the latest Chrome browser for Android and iOS can significantly reduce cellular data usage by using proxy servers hosted at Google to optimize website content [1]. For the purpose of our experiments, we did not enable this feature. Another example is streaming music using iTunes. At various time intervals, iTunes stops streaming music for commercials which are accompanied by graphical updates. When we encountered this case, we discarded the results.

Based on the above list of challenges, it is evident that comparing the energy-efficiency of mobile platforms/apps is a useful but challenging task. Therefore, new cross platform tools are needed in order to increase the accuracy of such comparison. In addition, an exhaustive list of rules for accurate data collection and procedure for energy-efficiency comparison is needed in order to avoid measurement errors.

3. EXPERIMENTAL APPROACH

Toward achieving our goal of comparing the energy efficiency of platforms and apps, we employ devices of various form factors from all three major mobile platforms, namely, Windows, iOS, and Android, for energy characterization as shown in Table 1. This section summaries the tools we use for energy characterization for these devices.

3.1 Windows

We used Intel SoC Watch for Windows [2]. We collected the following: 1) CPU idle sleep states for the package and cores. 2) CPU frequency. 3) Number of wakeups. 4) Timer resolution intervals. 5) Number of threads per application.

EnergyMeter: An Energy Profiling Tool for Windows. We developed *EnergyMeter* which takes as an input the test duration and outputs the energy consumed by the platform, package, cores, and GPU in joules. For platform energy consumption, we relied on Windows API to get a handler to the battery in order to collect the delta of the battery capacity changes for the test duration.
In order to collect package, core, and GPU energy consumption, we relied on hardware counters. The processor supports four Machine Specific Registers (MSRs) for Running Average Power Limit [3]. *RAPL_POWER_UNIT* reports power, energy status, and time units. *PKG_ENERGY_STATUS*, *PPO_ENERGY_STATUS*, and *PP1_ENERGY_STATUS*, report package, core, and graphics energy consumption. In order to calculate the energy used by each component, we calculate the ΔE_{MSR} and multiply it by the energy unit obtained from *RAPL_POWER_UNIT*.

3.2 Android

In order to power profile our Android device, we used the Trepn profiler [4]. We were able to collect the following metrics: 1) CPU utilization per app. 2) Average power consumption and 3) virtual memory utilization per app. 4) Number of wakelocks, wifilocks, and threads per app. Due to the extensive overhead of Trepn, we used *SoftPowerMon* [5] to collect the CPU's idle sleep states and frequency. Please note that the overhead observed by Trepn did not impact the above metrics because the data collected are per app.

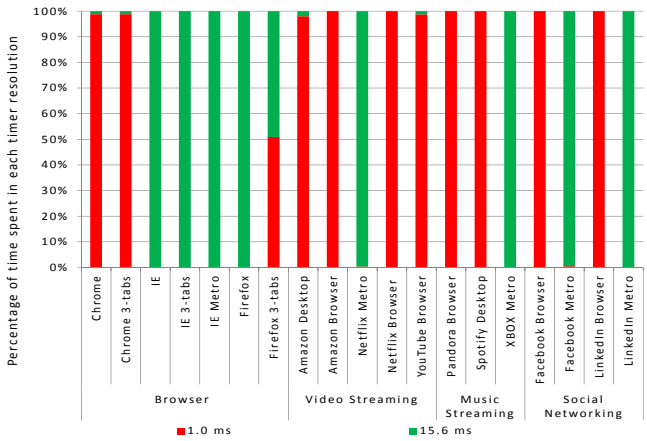

Figure 1: Timer resolution on Surface 2 Pro.

3.3 iOS

We used the *Energy Profiler Instrument* tool provided by Apple [6] and collected the following: 1) Energy consumption on an ascending scale from 0 to 20. 2) CPU utilization. 3) GPU utilization. 4) Total number of packets sent and received along with the total size in bytes for all packets.

Despite the fact that the above presented tools were capable of collecting some common metrics, however, each had different capabilities. SoC Watch and Trepn are capable of collecting a comprehensive list of metrics with fine grain precision. They are both capable of reporting per app values for several metrics such as CPU utilization. On the other hand, SoftPowerMon and the Energy Profiler Instrument are only capable of collecting a limited number of metrics with high coarse grain precision. They are also both incapable of reporting per app values for all of their metrics. Moreover, in terms of overhead, SoC Watch introduces 2% overhead which can reach up to 5% when the platform is highly active. Likewise, SoftPowerMon introduces 1-2% overhead. On the other hand, Trepen introduces ∼40% overhead even with small subset of metrics. Unfortunately, we were not able to measure the overhead introduced by Energy Instrument.

4. CASE STUDIES

Because comparing platforms and apps for energy efficiency in general is hard, if possible at all, we chose to use case studies as the first step to gain insights into comparing their relative energy efficiency. We selected four app categories and the corresponding list of most popular apps for each platform as shown in Table 2. Where applicable, we profiled the native and web-based app (Chrome was used for all web-based apps due to its availability on all three).

4.1 Browsers Scenario

We started profiling along a 3-minute timer, launched the browser (set to default webpage). Then upon the timer expiration, we stopped profiling and saved the results.

4.1.1 Surface 2 Pro Browsers

We profiled Chrome (1 and 3 tabs), IE (metro, 1 and 3 tabs), and Firefox (1 and 3 tabs) and ranked them as follows: Chrome, Chrome 3-tabs, IE, Firefox, IE 3-tabs, Firefox 3-tabs, IE Metro. ***Firefox 3-tabs vs. Chrome (Case***

Table 2: List of apps and corresponding version.

Scenario	Platform	App	Version
Browsers	Surface 2 Pro	Chrome	33.0.1750.146
		Internet Explorer 11	11.0.9600.16518
		Mozilla Firefox	24.0
	iPad Air	Chrome	32.1700.20
		Bing	2.0.2
		Safari	7.0.6
	Nexus 7	Chrome	33.0.1750.136
		Bing	4.2.3.20140303
		Mozilla Firefox	27.0
Video Streaming	Surface 2 Pro	Amazon Ubox Video	2.2.0.153
		Amazon (browser)	Feb 8, 14
		Netflix	2.3.0.12
		Netflix (browser)	Feb 8, 14
		YouTube (browser)	Feb 8, 14
	iPad Air	Amazon Instant Video	2.4
		Netflix	5.1.2
		YouTube	2.2.0
		YouTube (browser)	Feb 8, 14
	Nexus 7	YouTube	5.3.32
		YouTube (browser)	Feb 8, 14
		Netflix	3.2.1 build 1346
Music Streaming	Surface 2 Pro	Pandora (browser)	Feb 9, 14
		Spotify	0.9.7.16.g4b197456
		XBOX Music	2.2.444.0
	iPad Air	Spotify	0.9.3
		iTunes	7.0.6
		Pandora	5.2
	Nexus 7	Spotify	0.7.6.357
		Pandora	5.2
		Xbox Music	2.0.40226
Social Networking	Surface 2 Pro	LinkedIn HD	1.0.0.0
		LinkedIn (browser)	Feb 15, 14[1]
		Facebook	1.2.0.12
		Facebook (browser)	Feb 15, 14[2]
	iPad Air	LinkedIn	86
		LinkedIn (browser)	Feb 15, 14[3]
		Facebook	7.0
		Facebook (browser)	Feb 15, 14[4]
	Nexus 7	Facebook	6.0.0.28.28
		Facebook (browser)	Feb 15, 14[5]
		LinkedIn	3.3.1
		LinkedIn (browser)	Feb 15, 14

Table 3: Energy in joules on Surface Pro 2

Scenario	Application	Platform	Package	Core	GPU
Browsers	Chrome	734	148	22	0.69
	Chrome 3-tabs	763	165	34	1.2
	IE	824	177	36	4
	IE 3-tabs	853	247	111	0.89
	IE Metro	882	548	279	5.28
	Firefox	828	181	32	7.77
	Firefox 3-tabs	878	532	280	7.31
Video Streaming	Amazon Desk.	2023	635	135	39.83
	Amazon Browser	3063	1873	866	225.82
	Netflix Metro	1836	589	147	29.72
	Netflix Browser	3009	1493	496	259.14
	YouTube Browser	2476	1290	487	130.93
Music Streaming	Pandora Browser	1757	650	149	31.98
	Spotify Desk.	1598	404	65	0.3
	XBOX Metro	1465	332	57	3.5
Social Networking	Facebook Browser	853	165	23	1.37
	Facebook Metro	770	201	59	2.35
	LinkedIn Browser	799	160	32	1.17
	LinkedIn Metro	745	149	26	2.26

1): Chrome spent 99.8% in 1 ms timer resolution as shown in figure 1 significantly higher than Firefox 3-tabs causing the highest percentage of wakeups while having the same active duration as shown in Table 2. However, Chrome (1 and 3 tabs) still had a lower percentage of active cores and package compared to Firefox 3-tabs as shown in Figure 2. Thus, it was much more energy efficient. These counter intuitive results were justified once we examined the number of threads. Chrome had distributed its activities to seven threads, whereas Firefox only had one thread. As a result, Chrome took advantage of concurrency, which enabled both cores to go to sleep for longer duration and thus enabled the package to remain in idle sleep states for a long duration.

4.1.2 iPad Air Browsers

We profiled Chrome (1 and 3 tabs), Bing, Safari (1 and 3 tabs). The average energy levels are 3.87, 6.27, 2.65, 1.22, and 1.29 and CPU activities are 3.16%, 3.38%, 7.55%, 2.94%, and 3.29% for Chrome, Chrome 3-tabs, Bing, Safari, and Safari 3-tabs, respectively. They rank as follows: Safari, Safari 3-tabs, Bing, Chrome, and Chrome 3-tabs. One noteworthy observation is that Chrome consumed 62% more energy when we added two extra tabs, whereas Safari only consumed 5% more. ***Bing vs. Chrome (Case 2):*** Bing had the highest CPU utilization even though it is the second most efficient. By examining the network activities, we noticed the differences in communication patterns. In particular, Chrome received and sent large network packets after launching the browser 11,790.31, 4.91, and 12.18 KB in and 885.49, 3.23, and 3.56 KB out. Then, it sent and received small 80 bytes packets at an approximate 30-second intervals. On the other hand, Bing sent and received relatively smaller packets after launching the browser 260.9, 194.7, and 90.19 KB in and 11.79, 16.38, and 11.02 KB out. Then, received 60 bytes packets at an approximate 2-second interval. This example is counter intuitive because we expect that frequent communication reduces the energy efficiency of the platform. Unfortunately, the tool did not offer extra details in order to explain our observation.

Table 4: Metrics collected on Surface Pro 2.

Scenario	Application	Active Duration in ms.	Average Package Wakeups	Average Core Wakeups
Browsers	Chome	6,371	1015	1507
	Chrome 3-tabs	10,736	1061	1595
	IE	15,786	208	754
	IE 3-tabs	5,386	145	363
	IE Metro	10,647	316	1319
	Firefox	7,996	118	457
	Firefox 3-tabs	6,260	600	1141
Video Streaming	Amazon Desktop	75,638	1,094	1,900
	Amazon Browser	379,754	575	2,648
	Netflix Metro	28,750	393	1,019
	Netflix Browser	346,010	611	2,019
	YouTube Browser	361,663	622	1,628
Music Streaming	Pandora Browser	111,995	1,079	1,956
	Spotify Desktop	24,633	1,287	1,892
	XBOX Metro	18,774	286	567
Social Networking	Facebook Browser	9098	1,011	1,451
	Facebook Metro	19,618	260	551
	LinkedIn Browser	10,911	1,067	1,584
	LinkedIn Metro	15,420	232	569

Table 5: Metrics collected on Nexus 7.

Scenario	App Name	Ave. Power in mW	Ave. CPU Percent	Ave. Virtual Memory	Thr-ead Count	Total wake-locks
Browsers	Chrome	237	0.68	2966.63	88	1171
	Chrome 3-tabs	374	1.99	1989	66	1401
	Bing	745	7.04	912	20	0
	Firefox	235	0.09	1943	53	0
	Firefox 3-tabs	221	0.31	1955	52	0
Video Stream-ing	YouTube App	1,280	1.91	1003.22	65	0
	YouTube Browser	1,468	1.17	2146	77	1515
	Netflix App	1,387	3.04	1023.78	65	0
Music Stream-ing	Pandora	705	0.59	973.38	42	1714
	Spotify	683	4.01	1860.03	107	1646
	XBOX	483	2.08	981.23	45	1550
Social Network-ing	Facebook Browser	965	1.35	2089.94	72	1023
	Facebook App	1,211	1.74	1817.81	49	4
	LinkedIn Browser	640	0.55	2131.36	70	1024
	LinkedIn App	426	0.04	946.38	31	0

4.1.3 Nexus 7 Browsers

We profiled Chrome, Chrome 3-tabs, Bing, Firefox, and Firefox 3-tabs. We can rank them as follows: Firefox 3-tabs, Firefox, Chrome, Chrome 3-tabs, and Bing. ***Firefox 3-tabs vs. Firefox 1-tab (Case 3):*** It is odd to encounter this case where Firefox 3-tabs is more energy-efficient than 1-tab. One possible explanation is that the average CPU utilization

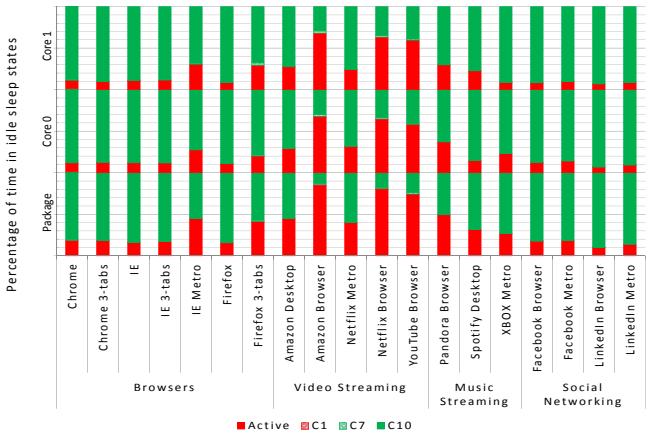

Figure 2: CPU idle sleep states on Surface 2 Pro.

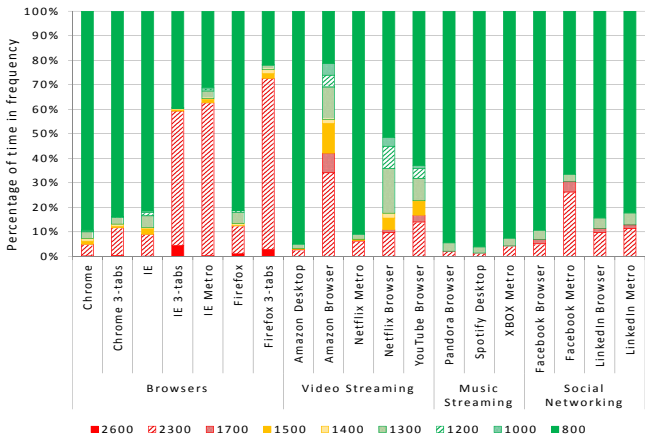

Figure 3: CPU frequency in MHz on Surface 2 Pro.

increased in the case of 3-tabs as shown in Table 5, leading to an increase in CPU frequency as shown in Figure 5 which lead to an increase in performance, which was translated to less core active duration as shown in Figure 4. ***Chrome vs. Bing (Case 4):*** Bing consumed more than triple the amount of power than Chrome as shown in Table 5. It may be attributed to the fact that Chrome has a higher multi-threading index than Bing.

4.2 Video Streaming Scenario

We started profiling along a 5-minute timer then played the video in full-screen mode until the timer expired. For the web-based case, we launched the browser and typed the credentials. Then, started profiling along a 5-minute timer. Next, we launched the browser and signed in. Then, we played the video in full-screen until the timer expired.

4.2.1 iPad Air Video Streaming

We profiled Amazon instant movies, Netflix, and YouTube (app and browser). The average energy levels are 10.73, 10.38, 10.47, and 10.71 for Amazon, Netflix, YouTube app, and YouTube browser, respectively. They rank as follows: Netflix, YouTube app, YouTube browser, and Amazon.

YouTube app vs. browser (Case 5): Even though streaming the same video using the app was more energy-efficient, however, using a browser had less percentage of CPU and graphics utilization. By examining the network activities, we noticed that YouTube app was constantly receiving packets. On the other hand, using a browser, led to much larger size of packets received at the beginning of the run (due to large buffering of the video), then throughout the test, there were long duration of 0 packet transmissions (20 seconds) followed by 5 seconds of activities. In theory, this method should enable the Wi-Fi radio to go to low-power states for an extended duration, thus reducing the energy consumption of the platform. However, buffering a large size of data led to more memory utilization, which nullified the savings from the sleep states of Wi-Fi radio and instead lead to an increase in energy consumption of the platform.

4.3 Music Streaming Scenario

We started streaming using an app (or browser). Then, we started the profiling tool with a 5-minute timer. Next, we relaunched the app (or browser) until the timer expired.

4.3.1 iPad Air Music Streaming

We profiled Spotify, iTunes, and Pandora and ranked them as iTunes, Spotify, and Pandora because the average energy levels are 5.35, 3.50, and 8.77 and CPU activities are 9.22%, 13.01%, and 9.18% for Spotify, iTunes, and Pandora respectively. ***Pandora vs. iTunes (Case 6):*** We noticed a reverse order of CPU utilization compared to the app energy-efficiency where the least energy-efficient app (Pandora) had the least CPU utilization, whereas, the most energy-efficient app (iTunes) had the most CPU utilization. Examining the network activity revealed that iTunes had sent and received during long time intervals large packets while at 2-second intervals received a small packet of 60 bytes. On the other hand, Pandora, sent out at regular 1-second intervals 166 bytes while sending and receiving during long time intervals large packets. Therefore, we can conclude that Pandora consumed more energy than iTunes because it kept the Wi-Fi radio at high power state for most of the test duration.

4.3.2 Nexus 7 Music Streaming

We profiled Pandora, Spotify, and XBOX music. ***XBOX vs. Spotify (Case 7):*** XBOX was the most energy-efficient but music streaming had many interrupts due to poor buffering. Therefore, XBOX sacrificed the user experience either to optimize the energy efficiency or due to poor app design.

4.4 Social Networking Scenario

We started profiling with a 3-minute timer. Launched the app. Upon the timer expiration, we stopped the collection. For the web-based app, we launched the browser and typed the credentials. Then, we started profiling along a 3-minutes timer. Next, we launched the browser and signed in. Upon the expiration of the timer, we stopped the collection.

4.4.1 Surface 2 Pro Social Networking

We profiled Facebook (browser and Metro), and LinkedIn (browser and Metro) and ranked them as follows: LinkedIn Metro, Facebook Metro, LinkedIn browser, and Facebook browser. ***Facebook Metro vs. browser (Case 8):*** The web-based version was slightly less active than the Metro version as shown in Figure 2 consuming less package and core energy as shown in Table 3. The platform energy consumption contradicted with package and core because the timer resolution was changed to 1 ms in the web based as shown in Figure 1. Thus, the average number of wakeups

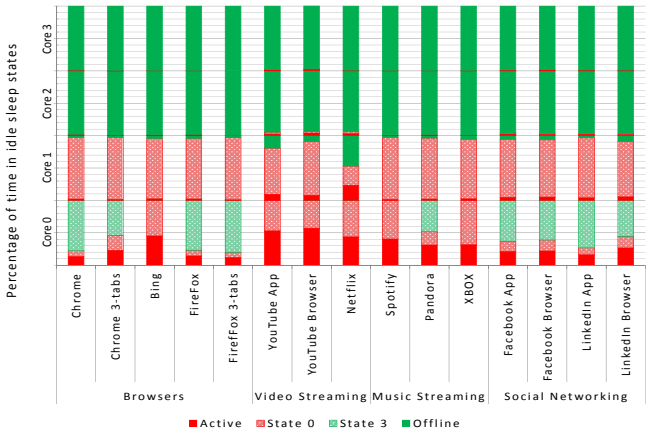

Figure 4: CPU idle sleep states on Nexus 7.

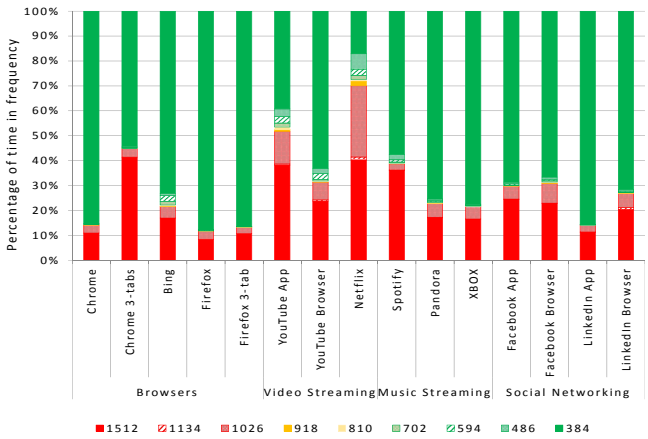

Figure 5: CPU frequency in MHz on Nexus 7.

was much larger than to Metro case as shown in Table 2. In addition, since the web-based version had lower resolution, the site got updated more frequently which kept the Wi-Fi radio active for longer duration and consumed more energy.

4.4.2 Nexus 7 Social Networking

We profiled Facebook (app and browser), and LinkedIn (app and browser) and ranked them as follows: LinkedIn app, LinkedIn browser, Facebook browser, and Facebook app. ***Facebook App vs. browser (Case 9):*** Unlike previous observations comparing app vs. web-based, the app consumed 25.5% more power than the web-based version as shown in Table 5. The possible cause may be attributed to the fact that the app had a sophisticated interface leading to 17% more GPU utilization than the web-based version.

5. INSIGHTS

Based on our analysis of the results, we deduced a list of insights and grouped them into the following four categories:

1- Power profiling mobile platforms and apps.

Designing and performing energy-efficiency comparison of platforms and apps are challenging tasks.

- Based on Section 2, new tools are needed in order to increase the accuracy of energy efficiency comparison of mobile platforms and apps. In addition, an exhaustive list of rules for accurate data collection and defined procedure are needed in order to avoid measurement errors.

- Debugging the energy efficiency of mobile platforms/apps is a complicated process where a particular power metric in a specific context can have a different meaning in another one. For instance, the higher the average wakeups value, the lower the energy-efficiency. However, that is not particularly true in the case where the average wakeups value is low but the percentage of CPU active duration is high. Therefore, creating a relationship model for correlating different metrics can significantly improve the debugging process.

2- The major three platform providers.

We observed that the three major platforms favor different tradeoffs between performance and energy efficiency.

- We noticed that apps released by Apple are more energy efficient compared to third party apps of the same category. One possible reason is that Apple's Energy Instrument had the least precise data collection options compared to other tools available on other platforms. As a result, we recommend enhancing the power profiling tool to collect additional power metrics with greater precision.

- We noticed that apps released by Google disregarded some of the energy-efficient principles in favor of performance. For example, Chrome browser on Windows changed the timer resolution to 1ms causing higher wakeups as shown in Case 1. In addition, Chrome browser on Android acquired high number of wakelocks as shown in Table 5.

- Based on our experiments, it seems that apps released by Microsoft were more concerned with energy efficiency than performance as shown in Case 8.

3- Energy-efficient app design.

We observed some app design strategies which are more energy-efficient than others.

- In general, native apps consume less energy than the web-based version as shown in Cases 5 and 8 reaching up to ∼11% in Case 8. Based on our data, we attribute the cause to the fact that native apps tend to have higher CPU utilization and lower memory utilization compared to the web-based counterpart. Charland *et al.* [7] discussed the strength and weaknesses of adopting both models. Combining their findings with the information presented in this paper can further help companies make educated decisions on the model to adopt. Moreover, based on our results, it seems that Google Chrome platform may be inherently at a disadvantage since it relies on web apps. Therefore, further related research is strongly needed.

- Despite the fact that buffering large data at a time can enable the Wi-Fi radio to go to an idle state, but it can also result in an increase of the power consumption as shown in Cases 2, 4 and 5. As a result, the size of the buffer needs to be balanced between the energy savings from enabling the Wi-Fi radio to go to an idle sleep state and the extra energy consumption due to the increase in memory usage.

- Multi threading increases the energy efficiency of an app if the execution is balanced across cores as shown in Cases 1 and 4. Sabharwal *et al.* [8] showed that if an app is multi threaded and balanced across the cores, then it enables the cores to work hard for a short duration, then enter a sleep state leading to improvement of energy efficiency. In addition, Carroll *et al.* [9] investigated how core offlining and DVFS can be used together in order to reduce energy consumption and developed Medusa, an offline-aware governor.

- Apps with low resource utilization (e.g., CPU utilization) but with high average wakeups can negatively impact the energy-efficiency of the platform as shown in Cases 6 and 8. High average wakeups of the platform's idle components

(e.g., CPU or Wi-Fi) results in switching the component from idle to active state leading to higher power consumption. For instance, in Case 6, iTunes average energy level was 3.5 whereas Pandora was 8.77 with network communication pattern being the obvious variable. As a result, timed interrupts and network communication should be coalescent in order to prevent unnecessary wakeups [10].

4- App developers practices.
Despite the vast number of tools and literature focusing on energy efficiency, we identified potential energy-efficiency improvements of some popular apps on all three platforms.

- We noticed that apps with the same functionality that are running on the same platform can vary vastly in terms of energy consumption (more than 50% in some cases as shown in Case 4). There are already several tools available for developers in order to power profile their apps. For instance, Kansal *et al.* [11] introduced an energy profiler which lets developers make power-aware design choices and trade off between energy consumption and performance of their apps. Another example is WattsOn [12] that estimates an app's energy consumption on the basis of empirically derived power models made available by either the smartphone manufacturer or mobile OS platform developers. These types of tools are very useful but there is also a need for power benchmark for each category of apps to be used as a baseline to compare the power consumption of apps instead of simply using the device's idle power consumption as the baseline.
- Changing the timer on Windows and holding wakelocks on Android seems to be common practices due to either lack of awareness of their power consumption overhead or a conscious decision to sacrifice efficiency in favor of performance.

The above list of insights summarizes our observations from comparing the energy efficiency of our categorized apps on all three platforms.

6. RELATED WORK

Jindal *et al.* [13] developed a taxonomy of sleep bugs in Android smartphones and categorized their root causes. Then, they used their model in order to evaluate 3596 APIs used in a set of 889 apps. Chen *et al.* [14] measured the energy drainage of the top 100 free apss in Google Play in order to determine the energy savings from prefetching ads. Wang *et al.* [15] used a collaborative approach to estimate the power consumption of mobile apps. They collected data from 120,000 Android users. Then, they used the data to build their power estimation model for mobile apps. These work focused on evaluating multiple single apps in order to energy profile them. We used the knowledge provided by their work in order to compare the relative energy-efficiency of apps belonging to the same category on multiple platforms. We also presented the challenges of such a comparison and showed that despite the research focusing on energy efficiency, still some of the popular apps on the three most popular platforms did not take full advantage of the available resources to improve the energy efficiency of their apps.

7. CONCLUSION

Energy efficiency comparison of mobile platforms and apps is a hard task due to possible measurement errors and challenges in designing case studies. Despite of the challenges, we compared the energy efficiency of four app categories: browsers, video and music streaming, and social networking on Windows, iOS, and Android. Based on the results, we derived a list of insights. In the future, we are planning on developing a power profiling analytical framework for developers in order to effectively power profile their apps.

Acknowledgment

This work is in part supported by NSF grant CNS-1205338. This material is based upon work supporting while serving at the National Science Foundation. We would like to thank our shepherd, Dr. Lin Zhong, for his constructive comments and suggestions in the preparation of the final version of this paper.

8. REFERENCES

[1] Google, "Data compression proxy," https://developer.chrome.com/multidevice/data-compression.
[2] Intel, "Intel SoC Watch for Windows," https://software.intel.com/sites/default/files/managed/aa/4a/socwatch_windows.pdf.
[3] Intel, "Intel 64 and IA-32 architectures software developers manual combined volumes: 1, 2A, 2B, 2C, 3A, 3B and 3C," http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf.
[4] Qualcomm, "Trepn profiler," https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler.
[5] G. Metri, A. Agrawal, R. Peri, M. Brockmeyer, and W. Shi, "A simplistic way for power profiling of mobile devices," in Proc. IEEE ICEAC, 2012.
[6] Apple, "About instruments," https://developer.apple.com/library/mac/documentation/developertools/conceptual/instrumentsuserguide/Introduction/Introduction.html.
[7] A. Charland and B. Leroux, "Mobile application development: web vs. native," Communications of the ACM, vol. 54, no. 5, 2011.
[8] M. Sabharwal, A. Agrawal, and G. Metri, "Enabling green IT through energy-aware software," IT Professional, 2013.
[9] A. Carroll and G. Heiser, "Mobile multicores: use them or waste them," in Proc. HotPower, 2013.
[10] B. Steigerwald and A. Agrawal, "Developing green software," Intel White Paper, 2011.
[11] A. Kansal and F. Zhao, "Fine-grained energy profiling for power-aware application design," ACM SIGMETRICS Performance Evaluation Review, vol. 36, no. 2, 2008.
[12] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in Proc. ACM MobiCom, 2012.
[13] A. Jindal, A. Pathak, Y. C. Hu, and S. Midkiff, "On death, taxes, and sleep disorder bugs in smartphones," in Proc. HotPower, 2013.
[14] X. Chen, A. Jindal, and Y. C. Hu, "How much energy can we save from prefetching ads?: energy drain analysis of top 100 apps," in Proc. HotPower, 2013.
[15] C. Wang, F. Yan, Y. Guo, and X. Chen, "Power estimation for mobile applications with profile-driven battery traces," in Proc. IEEE/ACM ISLPED, 2013.