How Much Energy Can We Save from Prefetching Ads? Energy Drain Analysis of Top 100 Apps

Xiaomeng Chen, Abhilash Jindal, Y. Charlie Hu
Purdue University

ABSTRACT

Recently, there has been a surge of interests on developing techniques and architectures for prefetching ads to potentially reduce the smartphone energy drain by 3G/4G radios from fetching ads. Despite the development of prefetching techniques, it remains unclear (1) how much smartphone energy do ads consume in popular apps in dominant app markets, and (2) out of which, what portion can we realistically save from prefetching?

We present a measurement study of the energy drain of top 100 free apps in Google Play, totaling more than 2.2 B downloads, to re-examine the above two motivational questions for ads energy research. We found the upper bound energy savings from prefetching ads is low: out of the top 100 apps, only 57 apps display ads, which incur on average 3.2% total energy on ads 3G tails. We further show the already-low upper bound ads energy saving is hard to achieve by ads prefetching as different apps exhibit very different ads behavior.

1. INTRODUCTION

Despite the incredible market penetration of smartphones and apps, the limited battery capacity has been and will remain a critical resource on smartphones that can seriously affect user experience. As such, optimizing the energy consumption of apps running on smartphones is of critical importance.

A major source of energy inefficiency of smartphones, which has been well known since at least 2009 [3], is the cellular radio, *i.e.,* 3G/4G, which exhibits a unique tail power behavior: the radio stays at a high power state for a long period of time after each active network activity, wasting a significant amount of energy.

This unique power behavior has motivated many studies in recent years on developing more efficient use of the 3G/4G radios in performing networking tasks. In particular, several recent work [9, 5, 6] studied saving 3G/4G energy due to ads which come with many free apps. Since ads are fetched and displayed periodically, fetching an ad involves a small transfer, and the long 3G/4G tail following each transfer can potentially lead to significant energy waste.

In [9], the authors analyzed the network trace collected at a major European mobile network and found ads account for 1% of all mobile traffic. They then proposed a background ads service named AdCache, that prefetches and caches ads to save energy. The study however did not evaluate the resulting energy savings on real apps.

In [5], the authors found the top 100 ads from AdMob are fetched on average 82 times each on each device, and proposed a middleware that predicts future user context and prefetches and caches ads when the phone is connected to a cheap and free network, *e.g.,* WiFi, rather than downloading ads on-demand over 3G/4G. The study focuses on saving user's monetary cost. The energy saving in the general scheme is minimal ($< 2\%$), since the ad module still has to contact the ad server for each ad slot, *e.g.,* to exchange the ad id, which is needed for online ad auctioning [6].

In [6], the authors found the top 15 ad-supported Windows Phone apps drain on average 23% of the total app energy. They recognized aggressively prefetching ads can lead to ads SLA violations and ad-server revenue loss, and proposed an ad proxy that learns the number of ad slots of each user phone in every hour and prefetches ads accordingly. However, its prediction accuracy is questionable; even the authors acknowledged that the app usage times, and hence the number of ad slots, are highly unpredictable when using user's history of each app's time of use.

Summarizing the state of affair, we have witnessed increasing interests on developing delicate techniques for prefetching ads to potentially save energy of apps spent on ads, yet it remains unclear (1) how much en-

HotPower'13 November 03-06 2013, Farmington, PA, USA

http://dx.doi.org/10.1145/2525526.2525848

ergy do ads consume out of the total energy drain of popular apps, and (2) out of which, what percentage can we realistically save from prefetching? The answers to these questions are important as they justify whether it is worthwhile expending efforts on developing sophisticated techniques for serving ads.

In this paper, we perform a detailed measurement study of the energy consumption of top 100 free apps in Google Play, totaling more than 2.2 B downloads, with the goal of answering the above two motivational questions for ads energy research. We found that out of the top 100 free apps, only 57 apps display ads, incurring on average 3.2% of total energy on ads 3G tails, and only 15 incur more than 5% total energy on ads 3G tails. These numbers constitute the upper-bound energy savings from eliminating ads 3G tails using an idealistic ads prefetching scheme. We uncovered a primary reason for this low energy saving upperbound, 3.2% on average among the 57 apps, is that a significant amount of ads energy, 13.2% on average, is already hidden behind app traffic or traffic due to other ad modules. We further provide evidence that these already low upperbound ads energy saving is hard to achieve by recent ads prefetching proposals as different apps have very different ads behavior; even the same ad module when used in different apps exhibit very different ads fetching behavior.

2. TAIL ENERGY BEHAVIOR

The tail power behavior of cellular radios, has been well known since at least 2009 [3, 8, 4]. For example, in a 3G network, the user equipment (UE) can be in the *idle* state, when it does not send or receive any data and the 3G radio draws nearly zero power, or the *FACH* state, when the UE performs low-rate transfer and the 3G radio consumes moderate power, or the *DCH* state when UE performs high rate transfer, which usually has its dedicated data channel and consumes high power. After a transmission, the radio goes through the transitions from DCH to FACH then to IDLE with specific timeouts, *e.g.,* 3s and 11s in an AT&T 3G network. The power drain and energy consumption during delayed transitions are known as tail power and tail energy. The long tail duration is controlled by the cellular provider, who tries to strike a balance between energy drain and responsiveness of the next network activity.

3. METHODOLOGY

3.1 Running Top 100 Apps in Google Play

We analyze the ads energy drain of the *top 100 grossing free apps*, *i.e.,* top 100 of all times, in Google Play. These apps are extremely popular with 100K+ to 100M+ downloads each. All the apps are run with typical user operations, *e.g.,* users read a few news feeds for news apps and make moves following game rules for gaming apps. To make our profiling runs representative, we carefully picked app run duration and user interactions.

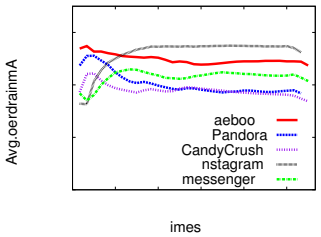


Fig. 1: Average power drain over time by top 5 apps.

Execution length. Figure 1 plots the average power consumed by the phone over time while running each of the 5 most popular apps. The results for other apps are similar and omitted for clarity.[1] We observe that while different apps show different transient behavior in the beginning, the average energy stabilizes for all of them after 5 minutes. Therefore, we run each app for 5 minutes in our energy profiling runs.

Execution variation. To capture variation in user interactions, all apps were run in two different modes – a *fast* mode where the user plays the app as fast as possible, and a *slow* mode where the user plays in a relaxed manner. All apps were run on a Samsung Galaxy SIII phone running Android 4.0 using AT&T 3G, and a Google Nexus One phone running Android 2.3 using T-mobile 3G. The results in terms of ads energy savings on the two phones very similar, and omit the Google Nexus One results due to page limit.

3.2 App/Ads Energy Profiling

We leverage the fine-grained energy profiler eprof [7] to perform energy profiling of the apps. Eprof breaks the energy consumption of an app profile run into that by each of the phone components (CPU, GPS, 3G, Wifi, sdcard), and by each of the app methods and threads. The energy estimation error of eprof was shown to be within 6% in profiling a set of diverse apps [7].

For our analysis, we need to further separate the energy portions due to ads, and due to the app itself. We make a key observation that apps typically use third-party ad modules, such as Google Admob, to display ads, and these ad modules almost always run in separate threads from the app's main threads. Therefore, to separate the energy drain due to ads and due to the app itself using eprof, we just need to identify all ad threads.

Separating app/ads threads. To identify ad threads, we first compile a list of most popular third-party ad modules from online sources [1], in particular, the *java*

[1]For simplicity, in this paper energy is presented in mAh (milli Ampere Hours). The actual energy value would be the mAh value multiplied by 3.7V, the battery voltage supply. This metric is used since smartphone batteries are rated using this metric and hence easy to correlate.

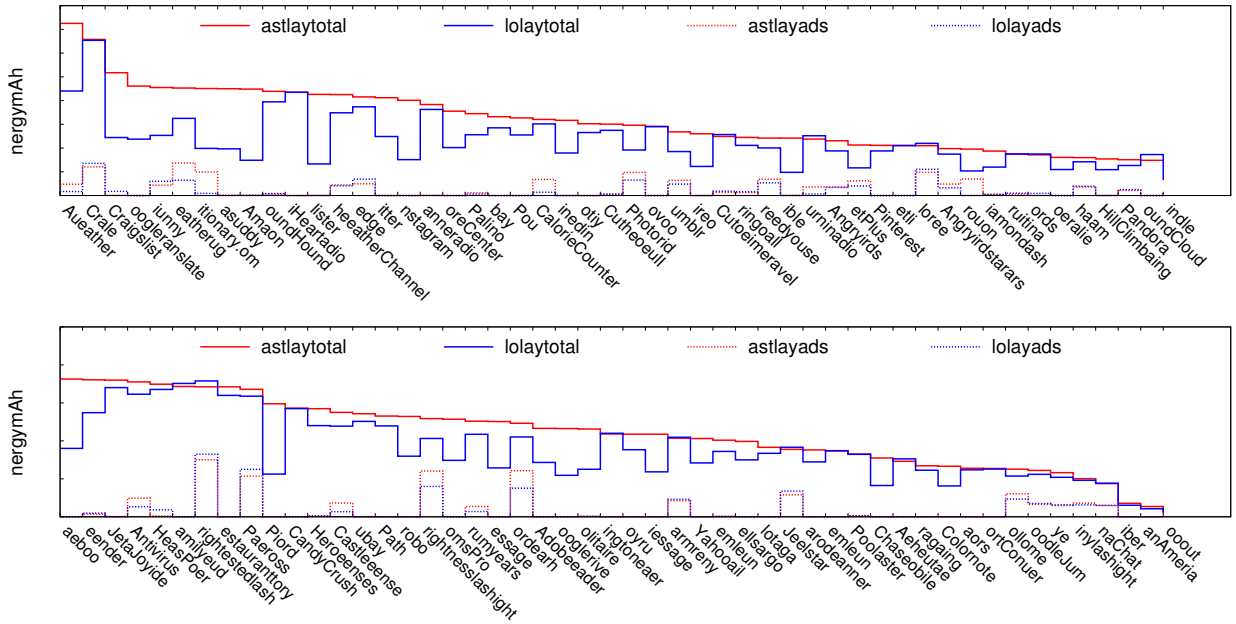


Fig. 2: Energy breakdown of top 100 free apps in Google Play on Samsung Galaxy S3. (Screen energy, 9 mAh for all apps, is not included for clarity.)

package names of these ad modules. Apps just include and appropriately invoke these ad java packages in order to display ads. Using this list of java package names, we can easily identify ad methods, and the threads in which these methods run are identified as ad threads. Finally, we label threads forked by ad threads as ad threads.

Energy accounting. Figure 3(a) shows a scenario where an app network activity led to 3G consuming E1 energy during transmission, followed by a truncated tail energy T1, due to an ads network activity, which consumes E2 energy during transmission, followed by a full tail energy T2. Eprof by default adopts a last-trigger policy, and would attribute E1+T1 to the app activity and E2+T2 to the ads activity. To perform the ads energy analysis in this paper, we adjusted the accounting policy to be app-first, where eprof first performs energy accounting by ignoring ads network activities, and then attributes the additional energy when taking ads network activities into account to the ads activities. Figure 3(b) shows under the app-first policy, E1 plus a full tail would be attributed to the app, and E2 plus the truncated tail equivalent would be attributed to the ads. Thus the app-first policy accurately reflects the fact if the ads were absent, the app would consume E1 plus a full tail.

4. HOW MANY APPS CONTAIN ADS?

Figures 2(a)–2(b) show the total energy consumption and ads energy consumption of the top 100 grossing free apps for both fast and slow runs, sorted by the total energy un fast mode in descending order. For apps allowing variation of user interaction speed, such as Amazon, Instagram and Firefox, the energy difference between fast and slow modes is large. For apps whose running speed can be barely impacted by user interactions, *e.g.,* music apps and games like Templerun, fast and slow modes do not result in much difference in energy drain.

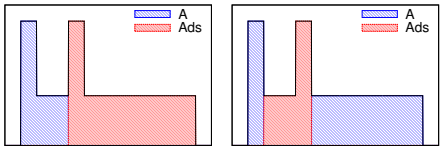


Fig. 3: (a) Eprof's last trigger accounting policy. (b) Refined accounting policy for ads.

We observe out of the top 100 apps, only 57 apps display ads using third-party ad modules. This shows that the common belief that free apps contain ads often does not hold. We further analyzed the rest 43 apps and found the following reasons why they do not show any ads:

Selling virtual goods Instead of showing ads, several gaming apps, *e.g.,* Temple Run and Candy Crush, sell virtual goods, and Flixster sells movies and TV shows, which are found to generate more revenue than ads [2].

Subscribers A large portion of popular apps, inevitably, consists of apps of the major services, such as Netflix, Dropbox and Bank of America. Such services already generate revenue from their subscribers; only the users with an *account* can use the app.

Self-generated ads Three apps, Facebook, Crackle, and

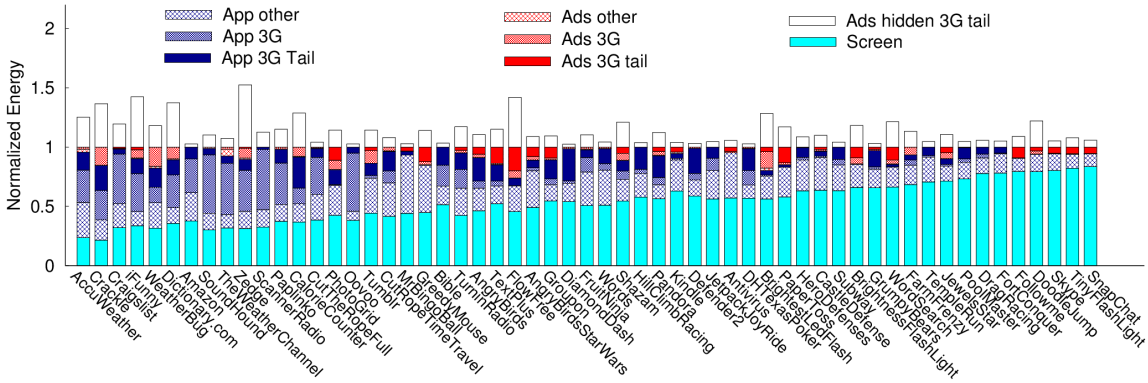


Fig. 4: Percentage energy breakdown of 57 top apps with ads in Google Play. (The app order in figure is the same as that in Figure 2(a)-2(b))

Pandora, generate their own ads rather than using third-party ad modules. Our methodology in §3.2 cannot separate such ads energy.

5. UPPER-BOUND ADS ENERGY SAVING

Next, we derive an upper-bound of how much ads energy can be saved for each of the 57 apps that show ads.

Methodology. To derive the upper-bound, we assume a perfect prefetching scheme that can accurately predict the exact number of ad slots that a user phone will have in each prefetching interval, *e.g.,* 30 minutes [6], and then uses an ad proxy similarly as in [6] to prefetch that many ads in one ad download. Prefetching this way effectively aggregates all ad downloads in a prefetching interval to share a single 3G tail. Since there can be multiple app runs in a prefetching interval, we further omit the energy cost of the aggregated download in calculating the upper-bound app energy saving. Effectively, the upper-bound ads energy saving for each app is calculated as simply the total 3G tail energy due to all the ad downloads in that app.

Results. Figure 4 shows screen energy and the 6-way normalized breakdown averaged over two runs of the total app energy into energy consumed by the 3G transfer, 3G tail, and the rest, due to the ads modules and due to the app itself, respectively, for the 57 top ads-displaying apps. The percentage ads 3G and 3G tail energy shown constitutes the upper-bound ads energy saving under perfect ads prefetching. Figure 5 shows the CDF of average upper-bound percentage energy savings for all top 100 apps. We observe that out of the 57 apps, 42 can save less than 5% energy, the remaining 15 can save between 5%–19.9%, with an average of 3.2% across the 57 apps from eliminating ads 3G tails.

Analysis. To gain insight into the limited ads energy drain in the top 100 apps under even perfect ad prefetching, we also plot the hidden ads 3G tail energy in Figure 4, calculated as the extra tail energy if the 3G network traffic of each ad module were not sharing any tails with the network traffic of the app or other ad modules.

Figure 4 shows the percentage hidden 3G ad energy ranges between 2.5%–52.4%, with an average of 13.2%. This is higher than the upper-bound ads energy saving, which ranges between 0–19.9%, with an average of 3.2%.

By studying the inter-play between app network traffic and ads traffic, we uncovered three typical scenarios where ads 3G tails are hidden behind other traffic.

Apps with intense network traffic hide ad 3G tails. Streaming apps, such as music apps, perform intensive data transfers which overlap with most of the ads traffic and hence hide their tails. Figure 6(a) shows the network traffic for the *SoundHound* app, where the ads 3G tails are well hidden by the app 3G traffic.

Ads and app data are fetched together. Several apps fetch ads on-demand, as opposed to with fixed update interval, but these demands are usually triggered by some user activity, which also lead to app network traffic. Such behavior is common in browsers, news apps, reader apps, *e.g.,* when a user opens a new page/news feed, the page is fetched and the ad is also refreshed. For such apps, the ad data also shares tails with app data. Figure 6(b) shows such alignment of app and ads traffic for the *Dictionary* app.

Ad modules hide each other's tail. Finally, apps typically use several ad modules (as many as 7!); the traffic from these separate ad modules can also share tail. Figure 6(c) shows the network traffic for the *Brightest flashlight free* app, where the app sets similar update intervals for both Millenial media and Adwhirl ad modules. As a result, they always share 3G tails.

6. UPPER-BOUND IS HARD TO ACHIEVE

To achieve the upper-bound energy savings shown in Figure 5 by prefetching ads, the prefetching system must accurately predict the number of ad slots the app will

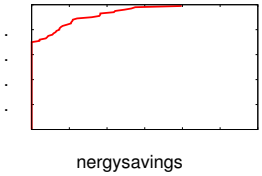
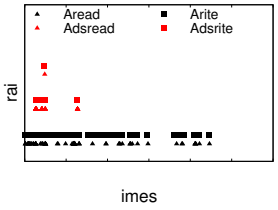
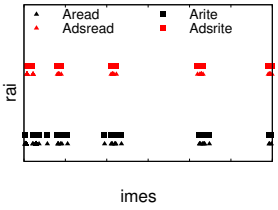


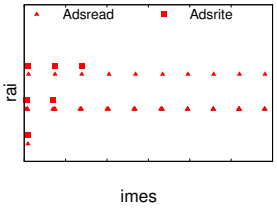
Fig. 5: CDF of upper-bound percentage energy savings for the top 100 apps averaged over slow and fast runs.



(a) Intense app network traffic (SoundHound)



(b) Synchronized app and ad network traffic (Dictionary)



(c) Synchronized ad fetching (Brightestflashfree)

Fig. 6: Ad 3G tail energy hidden by other network activities.

have for each prefetching interval, usually in the order of a few hours, *e.g.,* 1 hour in [6] to 2 hours in [9]. Accurately predicting the number of ads slots, however, is extremely challenging.

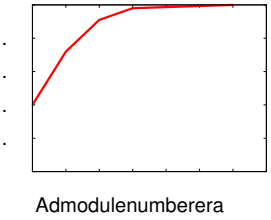
The authors of [6] make a simplifying assumption that all apps have the same ad fetching behavior – each app fetches 1 ad every 1 minute, and study how predictable is the number of ad slots per user phone in each prefetching interval, *e.g.,* 1 hour. Even under such a strong assumption, the authors find that the entropy of the number of ads in next one hour is very high, and acknowledge that the ad behavior of a user's phone is highly unpredictable (based on history).

Analyzing the 57 top apps displaying ads, we show any app-oblivious prediction is bound to give high prediction error for the following two reasons:

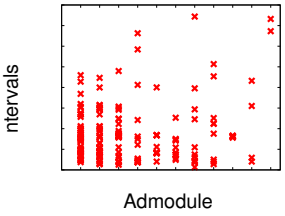
- Different apps are likely to have very different ads behavior, from using different ad modules which generally have different ads fetching behavior. Figure 7(a) shows that different apps use different numbers of ad modules. We found some apps to use up to 7 ad modules and 72% tested apps with ads use more than 2 ad modules. Further, ad modules Admob and Inmobi have different ad fetching behavior in terms of push vs. pull and supported refresh intervals [9].
- Moreover, even the same ad module when used in different apps can have different ad fetching behavior. Figure 7(b) shows that ad modules have different fetch intervals and one ad module in different apps have different fetch intervals. This is because some ads are fetched asynchronously where update intervals depend on some expected user operations, and the other ads are fetched periodically where the auto-refresh interval of ads is a configurable parameter set by the app developers.

7. CONCLUSIONS

We presented a measurement study of the energy consumption of top 100 free apps in Google Play, to re-examine the motivation for ads energy research: (1) how much energy do ads consume in popular apps in dominant app markets, and (2) out of which, what portion can we realistically save from prefetching? Our study shows out of the top 100 apps, only 6 incur more than 10% total energy on the ad modules. We uncovered a primary reason for this is that a significant amount of ads energy is already hidden behind app traffic or traffic due to other ad modules. Our findings suggest, in the absence of significant changes in ads behavior in the apps, there appears insufficient motivation for expending effort on developing sophisticated prefetching techniques.



(a) CDF of number of ad modules in each app.



(b) Update interval of 11 ad modules in 57 apps.

Fig. 7: Ad module usage variation across apps. In Figure 7(b), 1=mopub, 2=google, 3=millennialmedia, 4=admarvel, 5=inmobi,6=mdotm, 7=flurry, 8=jumptap, 9=adwhirl, 10=apsalar, 11=yume.

8. REFERENCES

[1] Andriod ad networks. www.appbrain.com/stats/libraries/ad.

[2] Revenue shift from advertising vs virtual goods sales. blog.flurry.com/Portals/41620/images/Flurry_AdvertisingRevenueShift_VGs-resized-600.png.

[3] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc of IMC*, 2009.

[4] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *Mobisys*, 2012.

[5] A. J. Khan, K. Jayarajah, D. Han, A. Misra, R. Balan, and S. Seshan. Cameo: A middleware for mobile advertisement delivery. In *ACM MobiSys*, 2013.

[6] P. Mohan, S. Nath, and O. Riva. Prefetching mobile ads: Can advertising systems afford it? In *EuroSys' 13*, 2013.

[7] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app? fine grained energy accounting on smartphones with eprof. In *Proc. of EuroSys*, 2012.

[8] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3g networks. In *Proc. of IMC*, 2010.

[9] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. Breaking for commercials: Characterizing mobile advertising. In *Proc. of IMC*, 2012.