Exploring the Effects of Ad Schemes on the Performance Cost of Mobile Phones

Cuiyun Gao*†, Jichuan Zeng*†, Federica Sarro‡, Michael R. Lyu*†, and Irwin King*†
*Shenzhen Research Institute, The Chinese University of Hong Kong, China
†Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China
‡Dept. of Computer Science, University College London, London, United Kingdom
{cygao, jczeng, lyu, king}@cse.cuhk.edu.hk, f.sarro@ucl.ac.uk

***Abstract*—Advertising is an important revenue source for mobile app development, especially for free apps. However, ads also carry costs to users. Displaying ads can interfere user experience, and lead to less user retention and reduced earnings ultimately. Although there are recent studies devoted to directly mitigating ad costs, for example, by reducing the battery or memory consumed, comprehensive analysis on ad embedded schemes (*e.g.*, ad sizes and ad providers) has rarely been conducted. In this paper, we focus on analyzing three types of performance cost, *i.e.*, cost of memory/CPU, traffic, and battery. We explore 12 ad schemes used in 104 popular Android apps and compare their performance consumption. We show that the performance costs of the ad schemes we analyzed are significantly different. We also summarize the ad schemes that would generate low resource cost to users. Our summary is endorsed by 37 experienced app developers we surveyed.**

***Index Terms*—In-app ads, performance cost, ad schemes**

I. Introduction

Advertising has experienced a tremendous growth recently, and has already become ubiquitous on mobile terminals. Organizations that have successfully monetized ad service enjoy huge profits. For example, for Facebook, mobile ad revenues raked in $9.16 billion in the second quarter of 2017 [10]. Compared with mobile web ads, in-app ads generally earn more user attention and ad revenue. According to a survey [11], mobile apps account for nearly 86% of the time spent on smartphones. However, underlying the ad benefits is the potential costs to users, such as battery drainage and traffic consumption. For example, ads can consume 23% of an app's total power [22]. There already exists research effort focusing on mitigating such costs, *e.g.*, prefetching ads to reduce battery drainage [22], or analyzing whether free apps cost users more money due to ads [17], [25]. Few studies analyze the effects of ad providers (*i.e.*, ad libraries) on users. For instance, Stevens *et al.* [27] examine the impact on user privacy of 13 popular Android ad providers by reviewing their use of permissions. Ruiz *et al.* [24] explore the relationship between ad providers and user ratings. However, comprehensive study on the effects of ad schemes[1] on performance costs of ads has rarely been implemented. Although Vallina *et al.* [28] discover that different ad service providers generate different traffic costs, they focus on characterizing the ad traffic. To fill the gap of existing research, we explore performance costs of different ad schemes, including ads' consumption of memory/CPU, traffic, and battery. We aim at assisting developers in determining more cost-effective and user-friendly ad schemes for better advertising.

The analysis on ad schemes includes several challenges. On one hand, ad schemes are not easy to be determined by manually interacting with apps, as different apps may involve ads with diverse ad sizes and ad providers. On the other hand, the cost of ad scheme is difficult to be separated from the costs generated by apps' intrinsic functionalities. To alleviate these threats in our paper, we initially employ static analysis on apps' decompiled code for capturing ad-related invocation, from which definitions of ad sizes[2] and providers are identified. Then we create a prototype app integrated with each ad scheme and improve the existing tool `IntelliAd` [15] for ad cost measurement.

In our study, we recognize 12 commonly-used ad schemes by analyzing 104 popular Android apps. Experimental results show that ad integration schemes can significantly impact the generated performance costs. Based on our findings, we suggest that developers choose ad service providers with low performance costs (*e.g.*, AdMob [4]). In terms of ad sizes, apps with full banners would generate relatively low performance costs. Moreover, to verify the practicality and usability of our suggestions, we surveyed 37 experienced app developers who have published one or more mobile apps.

The main contributions of this work are as below:

- We identify commonly-used ad schemes in 104 Android apps and are the first to comprehensively compare performance cost of different ad schemes as far as we know.
- We summarize the ad schemes that could produce lower performance cost. We provide developers with suggestions on designing cost-effective ad schemes.

The remainder of the paper is organized as follows. Section II describes the background and related work. Section III describes the framework we use for popular ad scheme acquisition and cost measurement mechanism. Our experimental

[1]An ad scheme include definitions of its ad sizes, ad service providers, the number of ads, etc.

[2]The definitions of ad sizes are for rendering ads in appropriate sizes for different orientations (portrait or landscape) and platforms (mobile and tablet).

study and lessons learned are presented in Section IV and V, respectively. Threats to the validity of our study are discussed in Section VI. Section VII concludes our paper.

II. Background and Related Work

A. Background

Generally, mobile ads are delivered in two types, namely in-app mobile ads and web ads. In this paper, we focus on the first delivery type due to its prevalence in people's daily life [9] and significance for benefiting app companies.

Advertising ecosystem of in-app ads comprises four major components: app developers, advertisers, ad service providers and end users. To embed ad contents in apps, developers typically need to register with a third-party mobile ad SDK, such as AdMob [4], MoPub [13], InMobi [12], etc. These ad SDKs are usually provided by ad service providers. The ad providers grant developers with specific ad controls, such as defining ad sizes. When rendering ad contents on end users' screens, apps send ad requests to the corresponding ad providers. The ad revenue for developers are calculated by the counts of displayed ads and clicked ads.

A recent survey [5] discovers that two in three mobile owners consider mobile ads annoying and want to uninstall the apps. These users may write poor reviews of the apps to express their complaints. For example, a one-star review of a fitness app states: "*It works, but it does things you can't stop, the ads suck up a lot of ram. Uninstalled due to ram usage, phone book modification.*". Such an unfavorable feedback can be referred by future potential users and lead to user loss [14]. Therefore, balancing users' dissatisfaction with ads and the ad benefits is crucial for app companies, and choosing cost-effective ad schemes is important to app developers.

B. Related Work on Ad Costs

Mobile ads can generate observable or unobservable costs for users and developers. For users, the costs can be related to mobile performance (*e.g.*, battery drainage), monetary loss (*e.g.*, traffic consumption), and potential security risk (*e.g.*, privacy leakage). Questionnaire-based survey is a canonical way for capturing user perceptions of mobile advertising [26], [30]. Many researchers also focus on alleviating ad costs. Mohan *et al.* [22] and Vallina *et al.* [28] develop a system for enabling energy-efficient ad delivery. Gui *et al.* [17] emphasize that the "free" nature in free apps comes with noticeable costs due to ads, including poor performance, traffic costs, and low user ratings. For developers, the cost is about app updates related to ads. According to Gui *et al.*'s work [17], 22% version updates have ad-related changes on average. Nath [23] discovers that the behavioral profiles collected by ad providers are not fully exploited, which may cause information leakage for users. To protect user privacy, Haddadi *et al.* [18] provide a networking protocol for targeted advertising. Gao *et al.* [15] create a tool `IntelliAd` for developers to automatically measure ads' consumption of memory, CPU, and traffic. Gui *et al.* [16] propose lightweight statistical approaches for measuring ad related energy consumption. To the best of our knowledge, no prior study has been conducted to comprehensively analyze the impact of different ad schemes on performance cost of mobile phones. A comprehensive survey on app store analysis can be found elsewhere [21].

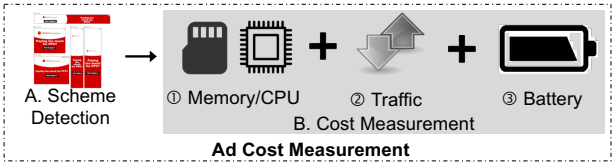


Fig. 1: Workflow for performance cost measurement of ads schemes.

```
import com.google.android.gms.ads.AdRequest;
import com.google.android.gms.ads.AdView;

public class MainActivity extends ActionBarActivity {
    ......
    AdView mAdView = (AdView) findViewById(R.id.adView);
    AdRequest adRequest = new AdRequest.Builder().build();
    mAdView.loadAd(adRequest);
}
```

Fig. 2: Code snippet for invoking an ad API.

III. Framework

The overall framework is outlined in Fig. 1. We first illustrate the method we use for capturing ad schemes integrated in popular apps (Section III-A). Then we introduce the method we use for cost measurement of each ad scheme (Section III-B).

A. Scheme Detection

To render ads, developers must instantiate specific class for invoking an ad API (*e.g.*, `com.google.android.gms.ads.AdView`) and define corresponding ad attributes (*e.g.*, ad sizes). So static analysis on source code can be conducted to identify embedded ad schemes. Since the source code of most popular apps is not publicly available, we employ `Apktool`[3] to decode apk files of these apps to nearly original code (smali code here). Due to numbers of ad service providers and various classes provided by these providers, capturing all ad-related classes automatically is difficult. Through observing the ad provider statistics provided by AppBrain [1], we discover that most developers use minority ad providers, *i.e.*, nearly 80% apps choose less than 20% ad providers. Thus, we focus on the top-20 ad providers (139 ad providers in total) and check whether the supporting classes are instantiated. Fig. 2 depicts an example code for invoking AdMob APIs. Once we detect ad invocation, we further analyze *layout* files or java source code to obtain the defined ad size.

B. Cost Measurement

Since our tool `IntelliAd` [15] covers the measurement of memory/CPU overhead and traffic usage, we only detail the measuring strategy of battery cost below.

[3]`Apktool` is a standard tool for reverse engineering Android apk files, available at http://ibotpeaches.github.io/Apktool/.

To efficiently measure battery consumption of each ad scheme, we leverage the framework `AppScope` [29]. `AppScope` comprises five components, namely CPU, LCD, WiFi, cellular and GPS. Since during experiment, LCD settings are consistent and GPS and cellular are switched off, we exclude these three factors from our cost measurement. Only the battery consumed by the CPU and WiFi components is considered for each scheme.

We quantify the battery consumed by WiFi based on packet rate, shown in Eq. 1, where p means the packet rate measured by *tcpdump* [6], β^{base} and β^{WiFi} represent coefficients, and t denotes the threshold for distinguishing packet rate in high frequency h and low frequency l.

$$P^{WiFi} = \begin{cases} \beta_l^{WiFi} \times p + \beta_l^{base}, & \text{if } p \leq t \\ \beta_h^{WiFi} \times p + \beta_h^{base}, & \text{if } p > t \end{cases} \quad (1)$$

To measure the power used by CPU, we utilize the average of CPU frequency, in which the CPU frequency can be measured through reading the utility */sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq*, shown as below:

$$P^{CPU} = \beta_{freq}^{CPU} \times u + \beta_{freq}^{idle}, \quad (2)$$

where $freq$ stands for average CPU frequency during runtime, and u denotes average CPU utilization recorded by *top* [7] ($0\% \leq u \leq 100\%$). β_{freq}^{CPU} and β_{freq}^{idle} are determined by using linear regression model based on the dataset provided by Yoon *et al.* [29]. Finally, the total power consumption is computed by combining the battery drainage on both components (*i.e.*, WiFi and CPU) as follows:

$$P = P^{WiFi} + P^{CPU}. \quad (3)$$

IV. Experimental Study

In this section, we elaborate on the subject apps and experimental settings, and result analysis. Our experiment involves 104 popular apps from Google Play in 2016, belonging to 19 categories (listed in Table I). The apps are top 100 apps of each category according to AndroidDrawer [8]. Table II summarizes the identified ad schemes of these subject apps, including the percentage, ad provider, ad size, and average user rating. We group the ad integration schemes into 12 groups based on ad sizes and ad providers identified from Section III-A. As shown in Table II, most subject apps (79.09%) are incorporated with ad APIs provided by AdMob, followed with MoPub (11.82%), and such distribution of ad service providers is roughly consistent with the ad provider statistics reported by AppBrain [1]. We aim at answering the following question:
RQ: Are there any significant differences among the performance costs of those ad integration schemes?

A. Experiment Setup

The experimental mobile device is an LG Nexus 5 smartphone with a rooted Android 5.0.1 operating system. We create 12 testing apps embedded with the ad schemes shown in Table II and one basic prototype app without ads. In the following, we explain the measurement strategies.

TABLE I: Subject mobile applications.

Category	# App	Category	# App
Business	4	Books & References	6
Comics	3	Education	2
Finance	2	Health & Fitness	15
Lifestyle	3	Media & Video	6
Medical	2	Music & Audio	6
News & Magazines	9	Personalization	6
Sports	1	Tools	1
Productivity	20	Social	6
Shopping	1	Photography	7
Weather	4	-	-

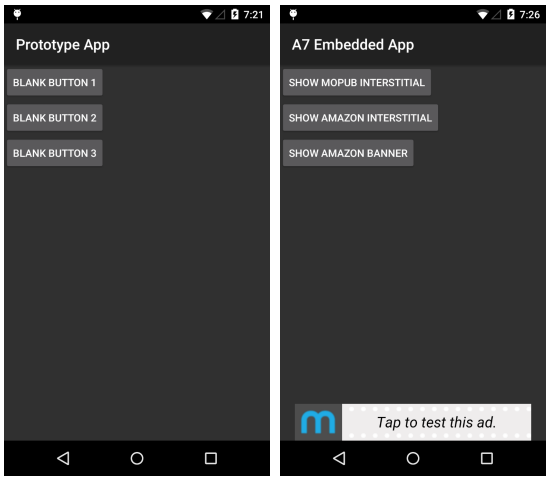


Fig. 3: The prototype app (left) and A7-embedded app (right).

Ad Number: According to mobile advertising policy [3], the number of banner ads on a single screen should be less than two. Therefore, for the ad schemes with more than one banner ad, we render them in separate activities. Since each scheme in Table II involves at most two banner ads, we implement two activities in our testing apps for rendering banners. For the schemes with only one banner, we leave one activity empty to ensure the number of activities is consistent.

App Design: The costs of ad schemes are measured by subtracting the costs of basic prototype app from those of ad-embedded apps. As Fig. 3 depicts, the prototype app (on the left) has three buttons, including two navigating to empty activities and the top one acting as a fake button. The fake button is utilized to render the interstitial ad in some schemes. The right screenshot presents an app *A7* with MoPub banner ad rendered. By clicking the three buttons from top to bottom, app screens display MoPub interstitial ad, Amazon interstitial ad, and Amazon banner ad, successively.

Source of Ad: According to existing policies [3], clicking on live ads is forbidden during app development and testing. Therefore, all the experiment ads are in their test mode.

Profiling Frequency: When an app is launched, `tcpdump` and `top` are started to profile real-time traffic usage and memory/CPU overhead. During intervals of two operations (set as 20s), the thread number and CPU frequency are captured by reading the system files every 0.04s. The average

TABLE II: Ad integration scheme summary.

ID	Ratio (%)	Ad Integration Scheme					Avg. Rating
		Ad Provider	Ad Size				
			Banner[1]	Smart Banner[2]	Full Banner[3]	Interstitial[4]	
A1	42.3	AdMob	✓				4.12
A2	13.5	AdMob	✓			✓	3.62
A3	12.5	AdMob		✓			4.34
A4	6.7	AdMob		✓		✓	4.14
A5	4.8	Amazon	✓				3.50
A6	4.8	MoPub	✓			✓	3.85
A7	3.8	MoPub Amazon	✓ ✓			✓ ✓	4.50
A8	3.8	AdMob			✓		4.15
A9	2.9	MoPub	✓				4.43
A10	2.9	AdMob				✓	4.07
A11	1.0	AdMob MoPub	 ✓	✓			4.70
A12	1.0	AdMob InMobi	 ✓	✓			4.70

[1] "Banner" refers to banner ads in standard sizes (320×50).
[2] "Smart Banner" represents ads with sizes that are self-tuned according to orientations (portrait or landscape) of mobile devices.
[3] "Full Banner" indicates full-size (468×60) banner ads.
[4] "Interstitial" refers to ads that cover whole interfaces of mobile devices.

values of these costs are considered for analysis. We measure four times for each scheme and take the average for analysis.

B. Ad Cost Analysis Results

In this part, we analyze the measured performance costs of each ad scheme in terms of memory/CPU, traffic and battery consumption.

1) Memory/CPU Overhead: The memory/CPU overhead is evaluated by three metrics [15]: memory consumed, CPU utilization, and thread numbers. For each ad scheme, we calculate its increase rates for these metrics by comparison with the basic prototype version, as shown in Fig. 4 (a).

The average increase rate for memory overhead and thread number is 1.17 times and 2.55 times, respectively. Even though CPU utilization presents the lowest growth rate (0.21%), it changes most obviously among the ad schemes (avg. stdev at 0.285). The average standard deviations for the costs of memory and thread are 0.185 and 0.248, respectively.

We further analyse the CPU utilization and find that its high standard deviation is mainly caused by the remarkable costs of A6, A7, A9, and A11. Since all these schemes integrate MoPub ads, we may attribute the high cost increase to the use of this ad SDK. We also discover that ad sizes influence CPU utilization. For example, although A1 (banner), A3 (smart banner), A8 (full banner), and A10 (interstitial) are all rendered with the same ad service provider (*i.e.*, AdMob), they display different performance for this metric. The increase rates of CPU utilization are 2.26%, 2.42%, 2.00%, and 3.40% for the four schemes respectively, which indicates that rendering interstitial produces the highest CPU overhead. For the schemes with different ad providers, such as A1 (AdMob banner), A5 (Amazon banner), and A9 (MoPub banner), their CPU utilization also varies. AdMob banner (2.3%) and Amazon banner (1.0%) definitively exhibit much better CPU performance than MoPub banner (32.8%).

Furthermore, the costs are also influenced by the number of ads (indicated by the number of check-off signs). For example, A7 with the most ads presents the highest increase rate (68.6%) of all for CPU utilization.

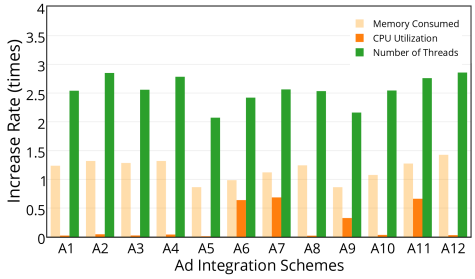
2) Traffic Usage: Traffic usage is measured by two metrics, namely total bytes transferred and packet numbers. Fig. 4 (b) depicts the growth rates for both metrics.

The average growth rates are 15.56 times and 4.29 times for the transmitted total bytes and packet numbers, respectively. Obviously, different ad schemes present different increase ranges. For example, apps with the AdMob SDK integrated (*e.g.*, A1-A4, A10-A12) exhibit more distinct increase than the other apps (*e.g.*, A5-A7). Regarding total bytes, A5 embedded with Amazon banner consumes the minimum traffic data (2.17 times increase) among all the banner ads, while the others (A1 with AdMob banner and A9 with MoPub banner) display more traffic consumption (9.63 times and 2.53 times respectively). Overall, ad schemes impact traffic usage in terms of ad providers and sizes.

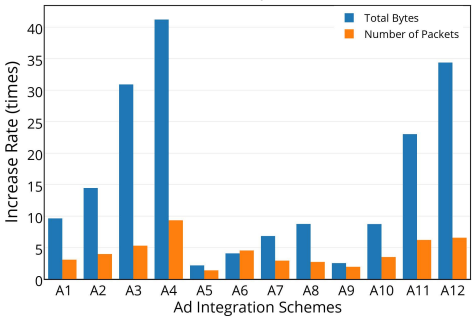
3) Battery Consumption: The method for measuring battery cost is explained in Section III-B, with increase rates of all the ad schemes shown in Fig. 4 (c).

All the ad schemes generate certain battery drainage. The average increase rate of consumed battery is 16.03% for the 12 schemes. Specifically, A6, A7, A9, and A11 present remarkable increase ratios than the other ad schemes, at 40.52%, 44.55%, 22.54%, and 46.12% respectively, and they all use the MoPub SDK. Therefore, we conjecture that the high battery consumption may be caused by the usage of this SDK. Also the performances of the schemes involving only one ad (*e.g.*, A1, A5, and A9) vary. For example, A5, embedded with Amazon banner ad, manifests the lowest cost (1.5%), while A7, embedded with the higher number of ads types, exhibits a higher power cost. Overall, we conclude that different ad integration schemes exhibit different performance on battery consumption.

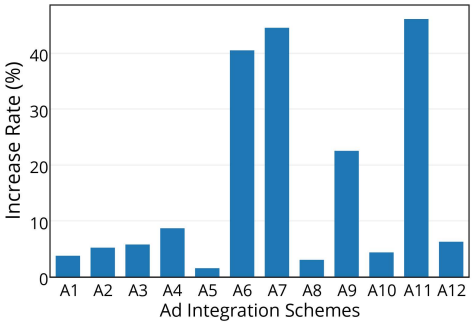
1) Overview of the Measured Costs: We then study which type of ad costs manifests the most significant distinction among the schemes we analyzed. Regarding memory/CPU



(a) Memory/CPU



(b) Traffic usage



(c) Battery

Fig. 4: Increase rate of memory/CPU (a), traffic usage (b) and battery (c) costs overhead for different ad schemes.

overhead, the average standard deviations of increase rates for the three metrics (*i.e.*, memory, CPU, and thread numbers) are 0.19, 0.29, and 0.25 respectively (shown in the left of Fig. 5). For the traffic usage, total bytes and packet numbers show standard deviations at 13.45 and 2.25, respectively (shown in the right of Fig. 5). This indicates that CPU utilization and total bytes present most salient variations in the corresponding cost types, and ad schemes can greatly affect CPU overhead and data bytes transmitted.

We then calculate Analysis of Variance (ANOVA) of the 12 schemes based on measured CPU utilization, packet numbers, and consumed battery corresponding to the three types of ad costs, and obtain the result of p-value at 0.0066 ($\ll$ 0.05), which demonstrates that the costs of different ad schemes are *significantly different*.

V. Lessons Learned

In this section, we discuss the lessons we have learned through our study. To further validate our findings, we have

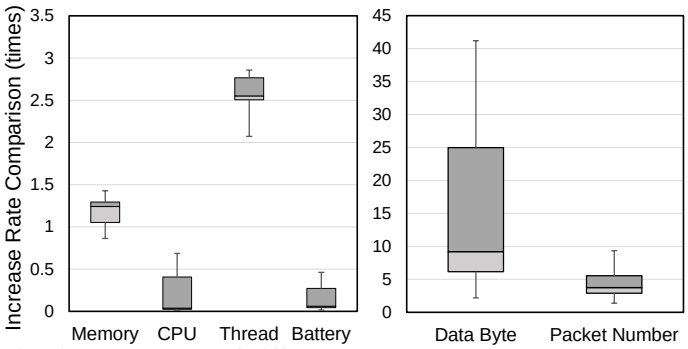


Fig. 5: Increase rate of different cost types over 12 ad schemes.

interviewed 37 experienced developers, who have created apps either for interest or for companies (*e.g.*, Google, Baidu, Tencent, TouTiao, and Ctrip, etc.), as done in previous work [20]. The interview was conducted by online questionnaire or conversation, and includes three main questions: One is about the number of apps they have created, one is for their concerns about in-app advertising during app development, and one for assessing the usefulness of our suggestions. All respondents have created one or more than one mobile app. Most of them highly agree that the lessons learned are helpful for embedding ads into apps. Specifically, all the developers surveyed think that our suggestions are meaningful, with 54.1% showing a strong agreement. 46.0% of developers state that they care more about the performance cost of ads, which demonstrates the importance of studying performance cost of ads. Besides the performance cost, the participants also provide other crucial issues of ad SDKs, such as security (10.8%), usability (8.1%), stability (5.4%) and validity (5.4%).

Developers are recommended to choose AdMob as ad provider: Most developers may employ ad SDKs provided by large companies or referring to ad payment. We focus on CPU utilization, which can be greatly influenced by ad schemes, and choose A1 (AdMob), A5 (Amazon), and A9 (MoPub) for analysis. As shown in Fig. 4 (a), AdMob (2.3%) and Amazon (1.0%) SDKs consume less CPU overhead than Mopub (32.8 %). Thus, AdMob and Amazon would be preferable to MoPub for developers. Considering that the proportion of apps with Amazon SDK (2.09%) is much less than that of AdMob (59.16%) [1], and higher user rating for A1-embedded apps (4.12) than A5-embedded apps (3.50), AdMob is highly recommended for mobile ad designing.

Developers are recommended to use full banner to display ads: One Google senior software engineer describes that suggestions on ad sizes and costs are considered very important in Google when devising apps. Here, we take A1 (banner), A3 (smart banner), A8 (full banner), and A10 (interstitial) as cases to study which ad size is more suitable. As shown in Section IV-B, A3 consumes the lowest memory/CPU, traffic and battery among the four types of ad schemes. Also, Table I shows that apps with interstitial ads have the lowest ratings (4.07), while the apps embedded with smart banners and full banners are rated higher (4.34 and 4.15, respectively).

Obviously, with interstitials displayed, users are prone to interact with them incidentally and feel irritated. Among all the banners, full banner seems to be a good choice for developers due to its relatively low performance cost and high rating.

VI. Threats to Validity

External Validity: First, our results are based on 104 apps from Google Play, representing an extremely small part of all the Android apps [19]. We alleviate this threat by ensuring that all the subject apps are popular apps listed by AndroidDrawer [8], an app release platform, and distributed into different categories. Also, our focus is to study the costs of ad schemes. The subject schemes are representative for popular ads design, as their rankings are roughly consistent with ad provider statistics [2]. Secondly, we investigate only Android apps from Google Play, so it is uncertain whether our suggestions are applicable to other mobile ads in other stores (*e.g.*, App Store or Amazon Store). However, since ad rendering mechanisms are similar in app markets, our suggestions may work also for others.

Internal Validity: First, we use prototype apps instead of real apps for measuring ad costs, which might bring some bias to cost measurement of ad schemes in real apps. As the source code of subject apps is not available and removing ads completely from smali code of popular apps is difficult and time-consuming, we choose not to separate ads from the more than 100 subjects. Using prototype apps [15] is an appropriate and efficient way to measure performance costs of different ad schemes. Second, we do not consider other ad formats, such as video ads and ad placement. We only explore ads in image in this paper, and leave video ads and analysis on more ad attributes for future work. Third, rendering duration of ads may affect the costs measured. Since we employ dynamic analysis to monitor the whole process, each ad has a similar displaying period. Utilizing the average cost per second would alleviate this threat, and the impact of rendering duration on our analysis results could be neglected. Also, to ensure the reliability of our experiments, we repeat the cost measurements of each scheme four times and take the average for analysis.

VII. Conclusion

In this paper, we have measured and compared the performance cost of different ad schemes. We have observed that there exist significant differences among different ad schemes, and developers should choose appropriate ad providers and sizes when embedding ads into their apps. In future work we will explore more attributes of ad schemes, *e.g.*, displaying period and refresh rate, and involve a large number of real apps for cost analysis.

Acknowledgment

The work was fully supported by Microsoft Research Asia (2018 Microsoft Research Asia Collaborative Research Award), the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14210717 of the General Research Fund), and the National Natural Science Foundation of China (No. 61332010, 61472338).

References

[1] Ad libraries provided by AppBrain. https://bit.ly/2K4FPMo.
[2] Ad networks and publishers list. https://bit.ly/1PSzQpb.
[3] Ad policies. https://goo.gl/afYFFe.
[4] AdMob. https://www.google.com/admob/.
[5] Ads dislike by users. https://bit.ly/2ahFPct.
[6] Android tcpdump. https://www.androidtcpdump.com.
[7] Android top. http://blog.djodjo.org/?p=349.
[8] AndroidDrawer. http://www.androiddrawer.com/.
[9] App vs. mobile web battle. http://venturebeat.com/2015/07/28/.
[10] Facebook ad revenue. https://bit.ly/2w3WUQ7.
[11] In-app ads. https://bit.ly/2yJmu2g.
[12] InMobi. http://china.inmobi.com/.
[13] MoPub. http://www.mopub.com/.
[14] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang. Investigating the relationship between price, rating, and popularity in the blackberry world app store. *Information and Software Technology*, 87:119 – 139, 2017.
[15] C. Gao, Y. Man, H. Xu, J. Zhu, Y. Zhou, and M. R. Lyu. Intelliad: assisting mobile app developers in measuring ad costs automatically. In *Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 253–255, 2017.
[16] J. Gui, D. Li, M. Wan, and W. G. J. Halfond. Lightweight measurement and estimation of mobile ad energy consumption. In *Proceedings of the 5th International Workshop on Green and Sustainable Software, GREENS'16*, pages 1–7, 2016.
[17] J. Gui, S. Mcilroy, M. Nagappan, and W. G. Halfond. Truth in advertising: The hidden cost of mobile ads for software developers. In *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*, pages 100–110. IEEE, 2015.
[18] H. Haddadi, P. Hui, and I. Brown. Mobiad: private and scalable mobile advertising. In *Proceedings of the 5th ACM international workshop on Mobility in the evolving internet architecture*, pages 33–38. ACM, 2010.
[19] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang. The app sampling problem for app store mining. In *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR'15*, pages 123–133, 2015.
[20] W. Martin, F. Sarro, and M. Harman. Causal impact analysis for app releases in google play. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE'16, pages 435–446, New York, NY, USA, 2016. ACM.
[21] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. A survey of app store analysis for software engineering. *IEEE Transactions on Software Engineering*, 43(9):817–847, 2017.
[22] P. Mohan, S. Nath, and O. Riva. Prefetching mobile ads: Can advertising systems afford it? In *Proceedings of the 8th European Conference on Computer Systems*, pages 267–280. ACM, 2013.
[23] S. Nath. Madscope: Characterizing mobile in-app targeted ads. In *Proceedings of the 13th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 59–73. ACM, 2015.
[24] I. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. Hassan. On the relationship between the number of ad libraries in an android app and its rating. *IEEE Software*, 99(1), 2014.
[25] R. Saborido, F. Khomh, G. Antoniol, and Y. Guéhéneuc. Comprehension of ads-supported and paid android applications: are they different? In *Proceedings of the 25th International Conference on Program Comprehension, ICPC*, pages 143–153, 2017.
[26] S. Soroa-Koury and K. C. Yang. Factors affecting consumers responses to mobile advertising from a aocial norm theoretical perspective. *Telematics and Informatics*, 27(1):103–113, 2010.
[27] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen. Investigating user privacy in android ad libraries. In *Workshop on Mobile Security Technologies (MoST)*, volume 10, 2012.
[28] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. Breaking for commercials: characterizing mobile advertising. In *Proceedings of Conference on Internet Measurement Conference (IMC)*, pages 343–356. ACM, 2012.
[29] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC)*, pages 387–400, 2012.
[30] J. H. Yu. You've got mobile ads! young consumers' responses to mobile ads with different types of interactivity. *International Journal of Mobile Marketing*, 8(1), 2013.