Dependability Issues of Android Games: A First Look via Software Analysis

Jiaojiao Fu*† Yangfan Zhou*† Yu Kang†‡

*School of Computer Science, Fudan University
†Shenzhen Key Laboratory of Rich Media Big Data Analytics and Applications
Shenzhen Research Institute, The Chinese University of Hong Kong
‡Dept. of Computer Science & Engineering, The Chinese University of Hong Kong
Email: jfu@cuhkri.org.cn, {yfzhou, ykang}@cse.cuhk.edu.hk

***Abstract*—Smartphones have surged into popularity in recent years, which has dramatically changed the way people live, work, and have fun. Smartphone games are an important type of Smartphone applications, which attract many software developers. However, they still have not caught much research attention in the software dependability community. In this paper, we study the characteristics of over 2000 Android games with software analysis techniques, with a focus on their dependability issues. Our study suggests that a new security paradigm is of great importance to Smartphone games to prevent potential privilege abuse. Moreover, a new set of testing and debugging approaches should be specifically tailored for Smartphone games, since games are becoming more complicated. Our study also reveals that most games are not specifically optimized according to the user pattern of Smartphones. We expect these open problems can bring more attentions to the software dependability community.**

I. Introduction

Recent years have witnessed the dramatic popularity of Smartphones, which have changed the way users live, work and have fun. Android is a prevalent open operating system for Smartphones [1], which recently becomes the most popular mobile operating system, sharing 79.3% of market shipments [2]. Among over 1 million applications (app in short) currently available in Google Play, the official Android app market, games form a typical type of apps which share over 50% [3]. Furthermore, according to Mobile Games Forum[4], games occupied the largest download times and more than 13 million users in EU (European Union) played games every day in 2012 [5]. The popularity of Smartphone games is even increasing in 2013 [6].

Although Smartphone games have quickly surged into popularity, there is still little research in the dependability community that focuses on this specific type of apps. This paper aims at providing the first look at the characteristics and new challenges introduced by Smartphone games in software dependability issues. To this end, we investigated over 2000 Android games with both static analysis approaches and dynamic ones. Our study reveals several important properties of the Smartphone games ecosystem that should bring attentions to the software dependability community. We highlight them in what follows.

1) A new security paradigm is of great importance to Smartphone games to prevent potential privilege abuse. We found that over half of the target games contain embedded advertisement modules (ads in short). Games typically request many privileged permissions for its ads such as accessing the Internet and the user location. These permissions can potentially cause privacy leak. A game with many privileged permissions is dangerous if being compromised. Hence, a new security paradigm should be designed for Smartphone games, for example, that can separate the ads which demand more privileged permissions from the native game codes with less privilege requirements.

2) Smartphone games are becoming more complicated, which pose great challenges to traditional testing and debugging approaches, especially for popular games that represent the trend of future mobile games. We found that 1) popular games tend to be larger in size, most of which (45% of the top 100 games) even require to install large separated data files in the extended storage (SD card typically) unlike typical Android apps that just install a stand-alone `.apk` file, 2) they generally require more privileged permissions and contain more embedded ads, and 3) they usually (84.4% of them) rely on a mixture of native C codes and Java codes to implement their functionality, unlike typical Android apps which are coded with Java only. Unfortunately, current testing and debugging methods generally cannot deal with such complications. Hence, a new set of testing and debugging approaches should be specifically tailored for Smartphone games.

3) Most games are not specifically optimized according to the user patterns of Smartphones. For example, when the games are just hanging in the background (typically due to the user pressing the Home button or just picking up a call), nearly half of them are still active in generating network traffic. This should generally be avoided: Most of the cases the game should be frozen since the user is not playing it. Continuous network transmissions may deteriorate device performance, drain battery power, and cause unnecessary resource usage (*e.g.*, 3G bandwidth). Game developers should be aware of the specific user patterns of Smartphones and optimize their apps accordingly.

The rest of the paper is organized as follows. Section II presents our experimental settings. In section III, we elaborate how we conduct our empirical analysis, what kinds of features we extract from each of the games, and how we obtain the feature data. We analyze the results obtained and provide implications on dependability issues of mobile games. The related work is discussed in Section IV. The paper is finally concluded in Section V.

II. Experimental setup

To study tremendous games currently available in Google Play, we resort to a sampling approach: We randomly download 1600 apps from an app market which are labeled as games. We consider these 1600 games can well reflect the characteristics of all available games in the market. We call these games the *randomly-sampled games* in our following discussions.

The popularity of a game well represents how people embrace it, which will consequently influence the Smartphone game industry. In other words, popular games can indicate the future direction of game design. In this regard, we also download the top 800 games according to their popularity, *i.e.*, the statistic of current installations. We call these games the *popular games* in our following discussions.

Each of the games is downloaded in Android application package file (APK) format. An APK file of an app contains all codes of the app, resources, assets, certificates, and a manifest file (*i.e.*, a configuration file) named AndroidManifest.xml.

We design a toolset, namely `Gamalysis`, to analyze the APK files. `Gamalysis` is an open-source java program, which contains several thousand lines of codes. The codes and our data set are available online at http://www.hkcloud.net/Gamalysis.

`Gamalysis` analyzes Smartphone games based on both static analysis and dynamic analysis. To conduct static analysis, `Gamalysis` first employs `apktool`, a tool for APK file decompilation and packaging [7]. It can then obtain the encapsulated contents of an APK file, *e.g.*, the bytecodes of the game for Dalvik virtual machine (where an Android app is running), and the manifest file AndroidManifest.xml. By parsing the bytecodes and the manifest file, `Gamalysis` can obtain a set of static features of a game.

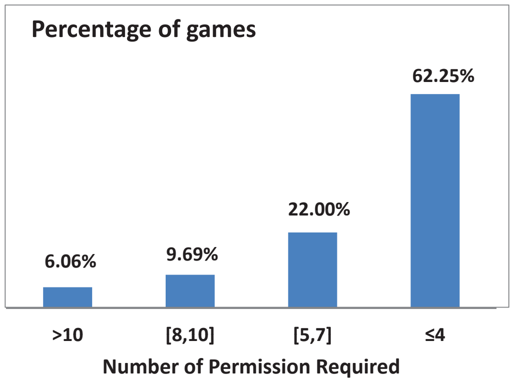
To conduct dynamic analysis, we use a real device (SAMSUNG S7562i). The device is connected to a computer using USB cable. `Gamalysis` relies on ADB (Android Debug Bridge) [8] to control the device. A target game can thus be installed, executed, and monitored (via an app we implement on Android) by `Gamalysis` without human intervention.

III. Empirical analysis: Methods and Results

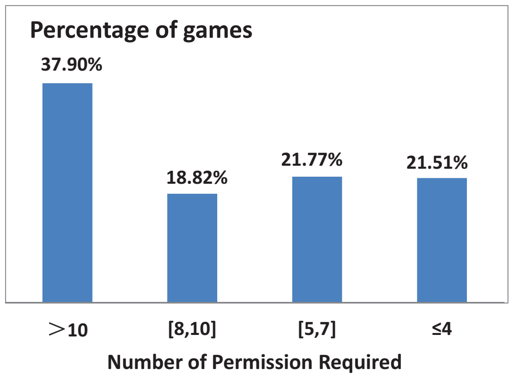
Now we discuss how `Gamalysis` extracts the features of a game app in terms of their permission requests, ads included, NDK usage, and Internet traffic, and provide the results of our empirical analysis, respectively in what follows.

A. *Privileges to access system-protected resources*

Designed as a privilege-separated operating system (OS), Android executes each app with a distinct system identity, which can isolate apps from each other and from the OS. Thus, each app can have its own privilege to access the system resources, which is provided through a permission mechanism. With a particular permission, an app can access the resource associates with the permission.



(a) Randomly-sampled games



(b) Popular games

Fig. 1. Distribution of permission numbers

Each app can claim its required permissions in the AndroidManifest.xml file deployed with the app. When a user installs an app, she must accept all the permissions specified in the file. In other words, through analyzing the permission requirements in AndroidManifest.xml, we can know the privilege level of a game app. The permission requirements of a game reflect an important security feature of the game, which is hence of our interest for dependability analysis.

Each permission requirement is declared by a "uses-permission android" tag in AndroidManifest.xml. Therefore, `Gamalysis` employs apktool, a tool for APK file decompilation and packaging to extract the AndroidManifest.xml file of each target app, where the "uses-permission android" tags are analyzed and the permission requirements are thus obtained.

Figure 1 shows the distributions of the permission numbers that the target apps request. We can find from Figure 1 it is quite common that a large portions of the target games requests many permissions. Nearly 40% of the randomly-sampled games request over 5 permissions. The situation is even much more significant for popular games: 78.5% of the popular games request over 5 permissions, among which one even requests 37 permissions.

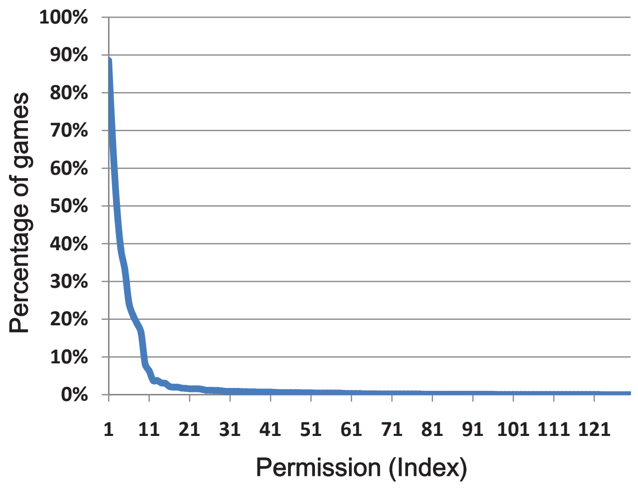


Fig. 2. Distribution of permission usage

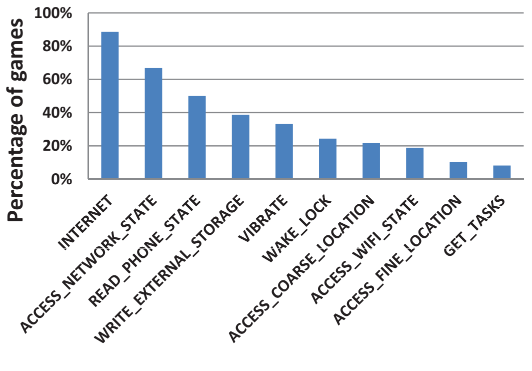
Figure 2 further shows how the 130 Android-defined permissions distribute among the target randomly-sampled games. We can see that only a few (ten in our experimental study) of the 130 permissions are frequently used by most applications and others are seldom used.

Figure 3 plots the detailed statistics of these 10 commonly used permissions for randomly-sampled games and popular games. We can see that `INTERNET` (*i.e.*, the permission to access the Internet) is the most frequently-used permission, which 88.4% of the randomly-sampled games require. 21.8% of the randomly-sampled games will obtain the user location. For popular games, the percentages are even larger. 97.8% of the popular games require the permission to access the Internet and 29% of them request to access the user location. Among these 10 permissions, there are 5 of them that can potentially allow a game to leak private user information or perform other malicious actions, which are listed in Figure 4.

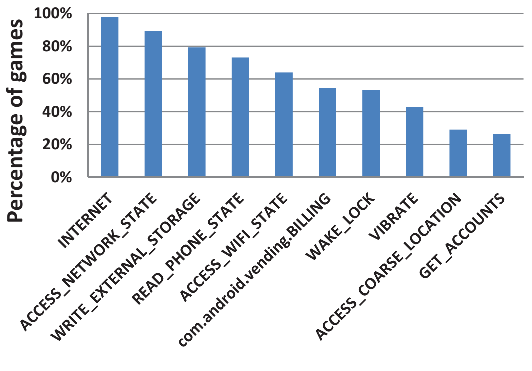
In general, an app with many privileged permissions is dangerous if being compromised. In this regard, the game which typically requires so many permissions is a potential security threat to Smartphones. A further investigation of why so many privileged permissions are required, and how to avoid a game to require so many privileged permissions in its design is of critical importance. Moreover, it is also interesting to examine whether a new runtime security paradigm is possible for Smartphone games to prevent permission abuse.

B. Advertisement modules

More than 73% of apps in Google Play are free. Apps usually contain third-party commercial advertisement modules (ads in short), which is a common way for app developers to gain revenue. To include third-party ads, developers have to bundle ad libraries from the ad service providers into their apps. Ads and game per se are providing different software functionality in nature, but they are packaged together and executed in the same process. Ads can inevitably influence the dependability of games. For example, Pathak *et al.* [9] found that ads spend 65%-75% of energy in free apps, which can deteriorate power efficiency. It is therefore important to study the ads in games.



(a) Randomly-sampled games



(b) Popular games

Fig. 3. Top ten permissions

`INTERNET`	Access the Internet
`READ_PHONE_STATE:`	Access sensitive phone information such as phone number and identity (IMEI)
`WRITE_EXTERNAL_STORAGE`	Write to external storage
`ACCESS_COARSE_LOCATION`	Obtain coarse user location
`ACCESS_FINE_LOCATION`	Obtain fine user location

Fig. 4. Example sensitive permissions

Ad library information is typically presented by app developer in AndroidManifest.xml. Figure 5 shows an example, where "com.google.ads.AdActivity" is the ad library and "ADMOD_PUBLISHER_ID" is the app identifier for the ad library.

Without specific information of an ad, we cannot know whether an app has included that ad. There are tremendous ad service providers, making it infeasible to identify each of them. Luckily, most apps with ads contain the most popular ads. For example, 99.7% of apps with ads contain the top 10 frequently used ad libraries [10]. Hence, we choose 13 popular ad libraries on Android platform and generated rules for identifying these libraries used by the apps, and disregard the rest of the ads, since they are rarely included.

Figure 6 presents the statistics of how many ads among the target 13 ads are included in games. We can see that over half of the games contain ads. Many games even contain more than 5 of the 13 most popular ad service providers. Popular games typically have more ads in one app than regular-sampled games.

```
<activity android:name="com.google.ads.AdActivity"
          android:configChanges="keyboard|keyboardHidden|orientation
                              |screenLayout|uiMode|screenSize|smallestScreenSize"/>
<activity android:name="com.facebook.LoginActivity"/>
<meta-data android:name="ADMOD_PUBLISHER_ID" android:value="a1518922cld2c5a"/>
<meta-data android:name="com.facebook.sdk.ApplicationId"
           android:value="@string/app_id"/>
```

Fig. 5. An example segment of AndroidManifest.xml with ad library information

Packaged together with ads, a game will execute in the same process with the ads. Therefore, a game will have to inherit the permissions required by the ads, and request them during the installation of the game. Figure 7 provides the permissions required by each of the 13 ads, where the first column shows the names of the ads. We can find that most ads need to perform privileged actions, *e.g.*, accessing the Internet and accessing sensitive phone information and locations. Hence, typically games have to request many privileged permissions for its ads, which can cause potential security problems, *e.g.*, privacy leak. New security paradigms can be helpful to enhance the security of Smartphone apps. For example, a paradigm which can separate the ads which demand more privileged permissions from the native game codes with less privilege requirements will help prevent potential privilege escalation by malicious codes.

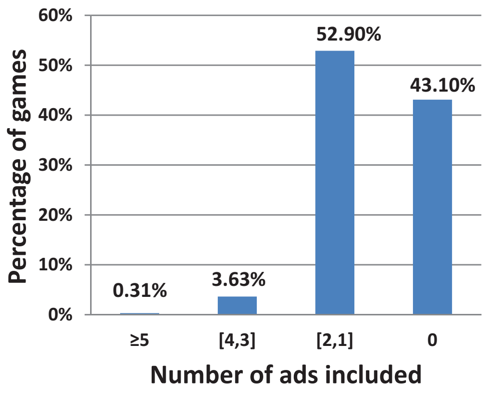
C. Background Internet traffic

More and more Smartphone apps rely on the Internet access to obtain remote data and provide richer functionalities. Game is not an exceptional case, as we have demonstrated that games typically requires the `INTERNET` permission. However, the Internet access is not free. The interface devices (*e.g.*, the WIFI transmission chip) can consume battery energy. Moreover, network bandwidth is limited resource shared by active apps. Unnecessary bandwidth consumption can drain battery power and deteriorate app performance, causing dependability problems. Therefore, we also study the Internet access of games. We employ a prevalent Internet packet capturing tool called `tcpdump` [11] to obtain the Internet access information of a target game.

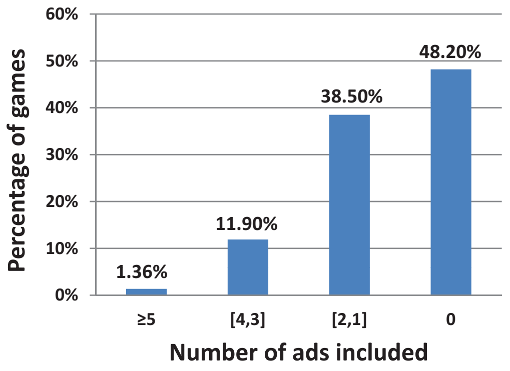
Since it is infeasible to play all our target games (over 2000 games in total) and analyze one by one the Internet access information, we resort to an automatic approach as follows. We implement an Android app, which can capture the Internet access of a target game by invoking `tcpdump`. The app is installed in our target device and run as a service in the background.

`Gamalysis` will automatically install a target game in the device, and then execute the game. The app will capture the game's Internet access information for one minute during which there is no human intervention. `Gamalysis` then simulates the scenario that the Home button is pressed by the user. This is a typical user pattern when the user wants to stop playing the game. It will generally cause the game app to run in the background. The app will also capture the Internet access information for 2 minutes. Finally, the game is terminated.

Our experimental results show that a game will typicall cause two kinds of the Internet traffic. One is for connecting



(a) Randomly-sampled games



(b) Popular games

Fig. 6. Distribution of the number of ads

a game server (*e.g*, that hosted in Amazon EC2 cloud) and the other for communicating with ad servers. One important phenomenon is that there are 48.87% among all target games that will still generate network traffic when they are running in the background.

Actually, most of the cases the game app is frozen since the user is not playing it. Internet traffic should meanwhile be avoided. Unnecessary network traffic may deteriorate device performance, drain battery power, and cause unnecessary resource usage (*e.g.*, 3G bandwidth).

When running in the background, active app behaviors (*e.g.*, communicating a remote server) are generally caused by app-defined Android service [1]. Game developers should be aware of the specific user patterns of Smartphones and accordingly optimize their apps, *e.g.*, freeze its unnecessary background services.

D. Popular games: The trend of Smartphone games

The popularity of a game reflects how people accept the game, which can also influence the game industry. In this regard, popular games can well represent the trend of future mobile games. Next, we discuss the features of popular game, comparing with the randomly-sampled games.

1) Size: First, popular games tends to be larger in size. Most popular games (45% of the top 100 in all target popular

	INTERNET	ACCESS_NETWORK_STATE	READ_PHIONE_STATE	ACCESS_WIFI_STATE	WRITE_EXTERNAL_STORAGE	ACCESS_COARSE_LOCATION	ACCESS_FINE_LOCATION	WAKE_LOCK	RECEIVE_BOOT_COMPLETED	RECORD_AUDIO	ACCESS_LOCATION_EXTRA_COMMANDS	READ_LOGS	GET_ACCOUNTS	READ_HISTORY_BOOKMARKS
google ads	√	√												
millennial media	√	√			√					⊙				
mobclix	√	√	√			⊙	⊙							
pontiflex	√	√	⊙			⊙	⊙						⊙	
airpush	√			⊙		⊙	⊙						⊙	⊙
mobfox	√		√											
greystripe	√	√												
youmi	√	√	√	√	√	⊙								
tapjoy	√	√	√	√		⊙	⊙							
inmobi	√	√										⊙		
madvertise	√					⊙	⊙							
leadbolt	√	√	√			⊙	⊙	⊙	⊙		⊙			
revmob	√		√	√										

Fig. 7. Permissions required by popular ads. $\surd$ and $\odot$ indicate the permission is a compulsory one and an optional one, respectively.

games and 30.9% of all) even require to install large separated data files in the extended storage (SD card typically) unlike typical Android apps that just install a stand-alone APK file. The reason is that the size of most popular games is huge. If being installed as a stand-alone APK file, all the contents will occupied the internal storage of a Smartphone. Hence, a game is divided into two parts, a relatively small `.apk` file and a set of large data files saved in the extended storage.

2) Permissions and ads: Second, popular games require more permissions and include more ads than randomly-sampled games do. As discussed in Section III-A, over 22% of the popular games request over 10 permissions, while only 6% of the randomly-sampled games request over 10 permissions. Figure 6 further shows that popular games generally include more ads.

3) Native Development Kit (NDK) usage: Android app is implemented with Java language, since Android provides a rich set of API libraries in Android Java SDK (Software Development Kit). In order to support C/C++ development, NDK (Native Development Kit) toolset is also provided, with which developers can implement part of app functionality in C/C++ in dynamic libraries, which can be automatically packed with Java codes into the same APK file [1].

It is commonly known that NDK is rarely used in general apps: Less than 10% of the apps used NDK [9]. In this regard, much research work on dependability focuses on approaches tailored only for apps implemented only with Java. The population of games that use NDK is an important consideration to tools for dependability enhancement. Hence, in this work, we investigate the popularity of NDK usage in games. We determine whether a target game uses NDK by examining whether dynamic libraries implemented with C/C++ (*i.e.*, .so files in general) have been packaged in its corresponding APK file. Our findings show that games usually rely on a mixture of native C codes and Java codes to implement their functionality, unlike typical Android apps which are coded with Java only. Popular games rely more heavily on NDK development (84.4%) than random-sampled games (5.8%).

4) A summary: We have shown that the trend of mobile games is that they are becoming larger and more complicated, which will involve more prelimited permissions and rely more heavily on a mixture of native C codes and Java codes to implement their functionality. Unfortunately, current testing and debugging methods generally cannot deal with such complications. Hence, a new set of testing and debugging approaches should be specifically tailored for Smartphone games.

IV. Related work

The popularity of Smartphones has attracted a large amount of research efforts. We survey the related work in this section.

Closely-related to this paper is the work that studies how to profile and analyze Smartphone apps. Wei *et al.* [12] profiled 27 apps from four layers: app specification, user interaction, operating system and network. Cho *et al.* [13] implemented a cross layer tool AndroScope, which is a performance tool for both Android platform and applications. Qian [14] designed ARO, the tool to expose the cross-layer interaction among various layers to discovery the inefficient resource usage for Smartphone apps. Specifically in Internet traffic analysis, Tongaonkar *et al.* [10] employed statistical analysis method to study the Internet usage behaviors of apps based on ad flows. Dai *et al.* [15] generated representative network traces and analyzed the network profiles of apps.

There are also many works studying the energy efficiency of Smartphones. Mittal *et al.* [16] presented an energy emulation tool empowering developers to estimate the Android apps energy consumption. eDoctor [17] is a practical tool that helps regular users to identify abnormal battery drain issues on Smartphones. Pathak *et al.* [9][18][19] and Jindal *et al.* [20] studied where the energy is used in an app and provide power models of different granularities to identify the energy bugs of Smartphones.

Besides, recent research has also focused on the specific applications. For example, Xu *et al.* [21] optimized email sync mechanism to decrease power consumption. LiKamWa [22] provided two mechanisms for improving energy efficiency of image sensing towards continues mobile vision. However, to our best knowledge, although games occupied almost half of the popular apps in software market like Google Play, there is no work focusing on Smartphone games anaysis.

V. Conclusion

This paper provides a first look at the characteristics of Smartphone games with a focus on their dependability issues on the Android OS. We study over 2000 games available in an Android app market. An analysis tool, namely, `Gamalysis` is developed, which can extract the features of a game app in terms of their permission requests, ads included, NDK usage, and Internet traffic. We conduct a statistical study on the data, and find out several dependability issues. The study suggests a call for research on a new toolkit for testing, debugging, securing, and optimizing Android games according to their specifics. We expect these open problems can bring more attentions to the software dependability community.

Future work of interest includes how to profile and analyze the energy efficiency of Android games, *i.e.*, the battery usage features of Android games.

Acknowledgement

This work was supported by the National Basic Research Program of China under 973 Project No. 2011CB302603, the National Natural Science Foundation of China under Project Nos. 61100077 and 61332010, and the Shenzhen Basic Research Program under Project No. JCYJ20120619152636275. Yangfan Zhou is the corresponding author.

References

[1] M. Butler, "Android: Changing the mobile landscape," *Pervasive Computing, IEEE*, vol. 10, no. 1, pp. 4–7, 2011.

[2] "Apple cedes market share in smartphone operating system market as android surges and windows phone gains, according to idc," http://www.idc.com/getdoc.jsp?containerId=prUS24257413.

[3] L. Aderemi, "Android as a gaming platform: Talk in the mobile game forum," 2012.

[4] Mobile Game Forum, http://www.mobilegamesforum.co.uk/.

[5] Z. Street, "Statistics from the mobile games forum," http://www.gamesbrief.com/2012/01/statistics-from-the-mobile-games-forum/, 2012.

[6] F. I. Younus, "What's new for mobile and virtual gaming this year," http://www.onislam.net/english/health-and-science/technology/461123-whats-new-for-mobile-and-virtual-gaming-this-year.html, 2013.

[7] apktool: A tool for reverse engineering Android apk files, https://code.google.com/p/android-apktool/.

[8] A. Developers, http://developer.android.com/tools/help/adb.html//.

[9] 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 *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 2012, pp. 29–42.

[10] A. Tongaonkar, S. Dai, A. Nucci, and D. Song, "Understanding mobile app usage patterns using in-app advertisements," in *Passive and Active Measurement*. Springer, 2013, pp. 63–72.

[11] TCPDUMP/LIBPCAP public repository, http://www.tcpdump.org//.

[12] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Profiledroid : Multi-layer profiling of android applications," in *Proc. of ACM MobiSys*, 2012.

[13] M. Cho, H. J. Lee, M. Kim, and S. W. Kim, "Androscope: An insightful performance analyzer for all software layers of the android-based systems," *ETRI Journal*, vol. 35, no. 2, pp. 259–269, Apr. 2013.

[14] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "Profiling resource usage for mobile apps: a cross-layer approach," in *Proc. of ACM MobiSys*, 2011.

[15] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "Networkprofiler: Towards automatic fingerprinting of android apps," in *Proceedings of the 32nd IEEE International Conference on Computer Communications, INFOCOM*, 2013.

[16] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM, 2012, pp. 317–328.

[17] X. Ma, P. Huang, X. Jin, P. Wang, S. Park, D. Shen, Y. Zhou, L. K. Saul, and G. M. Voelker, "edoctor: automatically diagnosing abnormal battery drain issues on smartphones," in *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2013, pp. 57–70.

[18] A. Pathak, Y. C. Hu, and M. Zhang, "Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011, p. 5.

[19] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 153–168.

[20] A. Jindal, A. Pathak, Y. C. Hu, and S. Midkiff, "Hypnos: understanding and treating sleep conflicts in smartphones," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 253–266.

[21] F. Xu, Y. Liu, T. Moscibroda, R. Chandra, L. Jin, Y. Zhang, and Q. Li, "Optimizing background email sync on smartphones," in *Proc. of the ACM MobiSys*, 2013, pp. 55–68.

[22] R. LiKamWa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl, "Energy characterization and optimization of image sensing toward continuous mobile vision." Mobisys, 2013.