International Journal of Computer Applications (0975 - 8887)
Volume 172 - No.9, August 2017

MobileMonkey - A Contextual Stress Testing Framework for Android Application

Rayhanur Rahman
Institute of Information Technology, University of Dhaka

Amit Seal Ami*
Institute of Information Technology, University of Dhaka

Kazi Sakib
Institute of Information Technology, University of Dhaka

ABSTRACT

Development of Android Apps is inherently challenging as difficulties arise in tracing bugs and crashes due to GUI based event driven work flow, contextual scenarios and diversified sources of inputs working together. In order to alleviate developer's challenges in this regard, a state of the art contextual stress testing framework of Android apps named $MobileMonkey$ is proposed. This framework facilitates developers to analyze Android apps using automatic stress inputs and contextual scenario generation with an inherent aim to invoke bugs or crashes, devised by a systematic and strategic execution of static analysis in a cohesive manner, which in essence, provides developers with plenty of insight regarding what went wrong based on data-intense crash logs, traceable trajectories of execution and replayable as well as replicable scripts. We evaluated MobileMonkeys effectiveness in comparison with industry standard Android app stress testing tool on 30 Android apps, 15 of which are heavily utilized real world android apps. The results demonstrate that MobileMonkey consistently performs better than the industry standard tool for stress testing in a diverse range of scenarios. Additionally, MobileMonkey is created to be resource friendly, horizontally scalable and non reliant on specific versions of Android Standard Development Kit, thus automatically becoming a better choice for being integrated as stress testing framework at any stage of Android app development.

Keywords

Android, Software Testing, Stress Testing, Contextual Testing

1. INTRODUCTION

The popularity of mobile devices and mobile applications is continuously growing due to utility and afford-ability. Due to usefulness, these are now touching critical domains such as NFC, SQUARE [28], health [17] and public services [30]. Google Play store, the official app store of the most popular smart devices platform, contains more than 2.7 million apps. Just for Android mobile devices, there were 2 billion monthly active users and 82 billion apps were downloaded from over 190 countries in 2016 [13]. Due to the large number of users, it is possible to earn by utilizing advertisements and other app related purchases. Considering its impact in the financial sector, Governments around the world are taking initiatives to encourage mobile app development by providing funds, initiating projects for enhancing skill related to mobile applications development [12]. However, the huge number of choices for apps makes the competition extremely high. To make the choices easier, Google Play Store displays statistics related to apps including the number of installations, average rating made by users based on Likert scale, ratings based on the answers of developer made questionnaire and user reviews. Consequently, the success of a new mobile app is dependent on User Reviews [18, 25], which are dependent on the usability of the mobile app itself.

Unfortunately, more than 1.2 billion Android apps in Google Play Store received less than 3 out of 5 rating by 2017 [8]. This means more than 40% of the apps were not satisfactory to the users. Even though Google Play Store do not provide guidelines related to app rejections; according to one of the dominant mobile app platform named Apple Play Store, the major dominant reason of app rejection is the lack of app completeness, or to elaborate in their words, *apps that crash or exhibit obvious technical problems* [9]. Irrespective of the fact that the number of mobile apps and demand for it are ever increasing, based on the number of low rated apps it can easily be determined that developers often do not prepare the apps completely. The key reason was found by Palit et al. [24] and established that newer software engineering approaches are required to test the mobile apps. This was further verified in the work by Muccini et al. [23], which showed that *mobile apps are different from traditional ones, and require different and specialized new testing techniques*. They further suggested that the key challenges are related to the contextual and always-mobile nature of mobile apps, such as sensed data provided by context providers, diversity of phone and phone makers and Graphical User Interface. This is an issue mobile app developers should be concerned about since only 16% of the users will try a bug-prone app more than twice [29].

In accordance with their suggestion and beyond, several state of the art works focused on providing contextual testing as a cloud service [19], providing testing as a service [15], modifying the Android SDK at the Java source level for generating input to unmodified Android apps [21], automated GUI testing [2], intent fuzzing [27], and automatic discovering, reporting and reproducing mobile app crashes through static and dynamic analysis [22]. Some of these works pushed the limits of mobile app testing and intro-

duced new concepts in terms of mobile app engineering. However, these approaches are likely to be provided as a commercial based service, will require submitting mobile app with/without source code and/or will be fuzzing the mobility related factors based on existing data. For example, utilizing a context library will definitely be dependent on submitted or collected contextual data [20], but it will also mean that it will not be able to provide contexts related to regions from where data was not collected. Some of the works are tightly coupled with Android SDK and underlying OS, thus making it unusable and less effective across different SDK versions and kernels. Moreover, an app developer from a developing country may not afford the services related to contextual testing as well, when just the instrumentation testing in virtual devices and real devices cost as much as much as $1 per hour and $5 per hour respectively [14]. This becomes a burden for outsourcing development related countries such as India, Brazil, China and Bangladesh where the hourly rate of app development starts from as low as $8 ($upwork.com$). Additionally, utilizing services requires mobile app developer to go through a *upload-test-view report-develop* cycle where development is dependent on the Internet. This introduces risk in the cycle as Internet disruption is still far too common and connectivity speed is questionable in developing and underdeveloped countries [1].

Addressing the aforementioned issues, MobileMonkey, a lightweight framework for contextual stress testing has been proposed that can work in parallel with/utilize existing industry tools such as Monkey [7], logcat [6] and Android Device Monitor [5]. In principle, it takes an Android mobile app installer in the form of $.apk$ file and tests it in emulator based on the user provided emulator OS and specifications, such as RAM size, number of processors and Android OS version. After that, MobileMonkey automatically analyzes the permissions requested by the $.apk$ file to determine suitable contextual scenarios for stress testing. Later, it produces contextual scenario trace and execution trace based on systematic static and dynamic analysis that can be utilized to reproduce the same contextual scenarios. User can provide custom contextual scenarios, or modify the produced contextual scenarios for further usage. Finally, it provides developers with comprehensive information regarding execution, crash with replayable and replicable scripts. MobileMonkey can work in parallel with instrumentation tests, but is not dependent on it. User can manually test or use the app on exploratory basis while MobileMonkey changes the contexts. The workflow of MobileMonkey is flexible enough to facilitate user intervention, instrumentation test and/or automated exploration by other tools.

The following contributions has been proposed through this paper:

(1) A developer friendly lightweight framework named MobileMonkey has been designed and implemented which can be utilized at any stage of app testing. The execution trace, along with the logs from logcat can be analyzed to deduce how the app behaved, whether it encountered any warnings, errors or failures. It is practical for the developer because it can run in parallel with instrumentation tests, exploratory tests or without user intervention. It is also unaffected by the different versions of Android Standard Development Kit (SDK) since it uses the APIs of SDK.

(2) It has been evaluated for 15 apps from the AndroTest repository. Since the apps are considered rather simple [33], it has been additionally evaluated with 15 Google Play Store apps with at least 4 out of 5 rating, over 1 million downloads and latest version released at 2017. The performance of MobileMonkey is compared with existing industry standard approach. The result shows that it is suitable for stress testing a mobile app by producing unexpected contextual scenarios.

2. RELATED WORK

Different concepts and approaches were introduced and evaluated in existing literature. The differences between mobile applications and traditional applications were discussed in [31] and concluded that user experience, non-functional requirements, portability and tools were areas that should software engineering research for mobile applications should focus on. Additionally, the differences were further investigated in [3] and concluded that reliable mobile application testing cost can be made cheaper through automation, and should consider the contextual and mobility nature of mobile applications. Several works are done to automate mobile application testing such as random input generation, code analysis based input generation and model based testing. In the following discussions, those approaches and novelty of MobileMonkey will be focused which surpasses limitations of those approaches.

At beginning, Random Input Generation based techniques gained momentum because of event based flow of process triggered by diversified source of inputs were the first line of testing strategies identified by the researchers. Such an approach is proposed in Dynodroid [21] where journal of event execution frequencies in a context-sensitive manner is maintained to trace corresponding events. Intent Fuzzer [27] conducts app testing by offline information extraction and static analysis with a pitfall of not being scalable for large apps. Vanarsena [26] is a Windows Phone testing approach based on analysis over app binaries and injection of random adverse contextual scenarios. The trend of random input generation is followed by more advanced systematic input generation techniques. These are mainly dependent upon heuristics such as BFS/DFS traversal on execution paths. For example, AndroidRipper [2] generates GUI based event triggering paths and profiles all possible permutation of events. Due to the extraordinary number of possible paths and scenario combinations, Convirt [20] introduced virtualized contexts by cloud enabled infrastructure for automated large scale mobile app testing based on static analysis.

However, even though these approaches are capable of stress testing an app to some extent, simply crashing an app is not much helpful if it is not traceable or does not provide enough information to find the underlying cause of such a crash. CrashScope [22] explored from this perspective and introduced a framework that can provide systematic input generation, crash detection, and replayable script related to crash. Several of the industrial approaches are also available for stress testing or contextual testing of Android app. For example, Google Test Cloud [15] and Xamarin Test Cloud [32] offers real device based infrastructure for app testing without contextual factors.

These approaches are compared against each other in different works from several types of perspectives. However, the comparative study concerning the application of these approaches in real world conditions showed that Monkey [7], a random input based stress testing tool that is available as part of the Android Standard Development Kit, performs better than the existing tools from the academia on real world apps. However, it also lacks the contextual event injection, replay-ability and configure-ability - thus making it

ill suited for stress testing mobile apps specially reliant on contexts.

In essence, all the aforementioned work provided app testing facilities for the developers with one or combination of GUI driven techniques, source analysis, binary analysis, SDK integration etc. However, there is still lack of testing facilities based on contextual information of specific mobility factors, lesser coupling with SDK integration and everlasting scope of performance improvements.

3. SYSTEM ARCHITECTURE

Algorithm 1 Overall algorithm of MobileMonkey

Input: Mobile App, Test Duration,
Optional: Minimum Interval, Maximum Interval, Seed, Contextual Events, Intervals, Steps
Output: List of Contextual Events
Ma := Mobile App
Td := Test Duration
MaI := Maximum Interval
MiI := Minimum Interval
S := Seed
Ce := Array(s) of Contextual Events
It := Array of Intervals St := Steps
INITIALIZATION:
1: Provide the Ma to MobileMonkey
2: Extract Permissions (P) requested by MA
3: **if** (St is $None$) **then**
4: $St = 0$
5: **end if**
6: **if** (St is $None$ and Ce is $None$) **then**
7: **repeat**
8: Based on S, generate array(s) of Ce from P at interval (i) between MiI and MaI until sum of all intervals is equal to Td
9: append i to It
10: $St = St + 1$
11: **until** ($St \leq Td$)
12: **end if**
EXECUTION:
13: Start Emulator
14: install app Ma
15: **for** $i = l$ to St **do**
16: execute $Ce[i]$
17: wait for $It[i]$ interval
18: uninstall Ma
19: Reset Emulator
20: **end for**

Based on the previous works, it can be deduced that a framework is necessary, which will be able to introduce contextual events in stress testing a mobile app. Furthermore, it will be configurable enough to automatically create scripts, as well as allow user to utilize custom scripts for stress testing app. The scripts generated should be replay-able and reproducible to ensure that the bugs are reproducible and traceable. Due to the regularly changing nature of Android Standard Developer Kit(SDK), naturally the framework requires less coupling and dependency of the SDK. Additionally, it should be resource friendly so that it can be utilized in standalone machines and horizontally scalable infrastructures alike.
Based on these conditions above, MobileMonkey has been schemed. In this section, the system architecture of MobileMonkey is presented. For easier understanding, an overall algorithm of MobileMonkey is provided in Algorithm 1. MobileMonkey is divided to several components, which are:

—Configuration Reader
—App Manager
—Contextual Events Generator
—Executor
—Log Analyzer

These components work independently and are reusable as separate components. For example, user can use manually generated contextual events in a specific format or modify generated contextual events. Those manually created events can later be used by Executor during stress testing instead of using the Contextual Events Generator.
The components are discussed in further details in the following subsections.

Table 1. Configuration of MobileMonkey

Configuration	Value
emulator	the name of the phone model to be used, such as Nexus6
emulator_port	the port using which the emulator will be connected, such as 5555
apk_full_path	the full path of the apk to be tested
uniform_interval	the uniform amount to be used in between contextual events
minimum_interval	the minimum interval to be used in between contextual events
maximum_interval	the maximum interval to be used in between contextual events
telnet_key	telnet key is used for low level communication with emulator, such as setting network delay
seed	the seed is used to generate pseudo random numbers. This ensures consistency of creating exact series of random series with the same seed.
duration	the duration for which MobileMonkey will continue testing
duration	the duration for which MobileMonkey will continue testing

3.1 Configuration Reader

Configuration reader determines the nature of MobileMonkey. It can configure itself by reading and assigning values, such as the emulator type, emulator OS type, the minimum and maximum interval to be used in between contextual events, the total duration of the execution of MobileMonkey, and the seed value to be used for randomization. The notable configurations used for MobileMonkey in key, value format is provided in Table 3.

3.2 App Manager

As the name implies, the App Manager manages the installation, removal and running of App. It also extracts permissions requested by the mobile app. These permissions can later be utilized by the Contextual Events Generator to generate relevant contextual events.

Table 2. Context Types and Possible Values of Mobile Monkey based on Android Guidelines [4]

Gsm Profile Strength	Network Delay	Network Status		Key Events	User Rotation
0	GSM	GSM	HSDPA	Alphabets	Potrait
1	EDGE	HSCSD	LTE	Numerals	Landscape
2	UMTS	GPRS	EVDO	Symbols	Reverse Potrait
3	None	UMTS	FULL	Backspace	Reverse Landscape
4		EDGE		Delete	

3.3 Contextual Events Generator

Based on the permissions extracted by the App Manager, Contextual Events Generator determines the types of tests it will use for the app. Scope has been limited to Network Speed and Delay, Airplane Mode, UI Rotation, GSM Profile and Keyboard events as contextual events. Among these, User Interface Rotation, Keyboard Events and Airplane mode are considered default contextual events. In MobileMonkey, values of context types are used on the basis of Android guidelines, which are provided in Table 3.3. Additionally, user will also provide the total duration of testing, minimum and maximum possible interval between each contextual event. Furthermore, a seed value is required by the internal randomness generator to maintain consistency similar to Monkey [7]. To illustrate, lets assume that for a mobile app called AppX user set the total duration as 10 seconds. After permission extraction from AppX, it is found that it utilizes `android.permission.INTERNET` permission. Contextual Events Generator will generate sets of contextual events related to `Network Speed`, `Network Delay`, `Key Events`, `User Rotation` and `Network Status`. Each set will contain relevant events with a randomly selected interval based on minimum and maximum interval. The total duration of intervals in a set will be equal to the total duration as specified by the user.These sets of contextual events are then fed to the MobileMonkey Executor, which injects contextual events to the Emulator while execution.

3.4 Executor

It is possible to facilitate manual, automatic and combined contextual stress testing while using MobileMonkey. To elaborate, a human user can perform regular operations such as tapping or changing orientation manually while MobileMonkey executes the generated contextual events. Furthermore, it is possible to execute more than one contextual event at the same time. For example, it is possible to have a Network Delay of GSM type, while having Network Status of HSDPA type at the same time. It should be mentioned that it generates logs similar to the application logs collected by logcat [6] for traceability. To illustrate, lets consider that it set Network Status to HSDPA type at a certain time. A MobileMonkey trace `06-17 18:53:29 NetworkStatus 1 10 hsdpa` will be generated, which describes that during execution, a Network Status of HSDPA was maintained for 10 seconds. It should be noted that this format is based on logcat's default logging format. To automate testing, MobileMonkey can also execute the Monkey [7] in parallel for the duration of the MobileMonkey. Monkey facilitates generating pseudo random input events, such as tapping, touching, gestures, and a few system events. It is possible to configure the nature of the Monkey, like interval between each events, number of events, and the seed to be used for random input event generation. Executor utilizes the settings provided by the Config Reader for the Monkey. An internal Fault Watcher monitors the app for Fatal Exceptions/Crashes and stops execution as soon as it is triggered. It should be clarified that MobileMonkey injects events independently, and it is also possible to use instrumentation tests on behalf of users while executing MobileMonkey contextual events. Lastly, after completing execution, the traces generated by MobileMonkey related to contextual events and Android OS are collected and sent to Log Analyzer for further analysis.

3.5 Log Analyzer

First, Log analyzer collects warning level logs and above from logcat. Next, it cleans the garbage data, misfired events, and non-relevant logs. Furthermore, it creates lists of warning, error and Fault incidents separately. Log analyzer also collects MobileMonkey traces regarding contextual scenarios. It then combines both traces to connect application logs with contextual scenarios. Consequently, it becomes easier to understand what type of contextual scenario helped create an unexpected application log. All the components and flow of MobileMonkey is depicted in Figure 1.
Due to its flexible architecture, MobileMonkey ensures low memory footprint and CPU utilization at any stage as well as allows to be used in conjunction with various approaches. For example,

—**As Service:** MobileMonkey can be utilized in a distributed system where it is utilized in various types of emulated OS models, and device emulator. This further helps stress testing a Mobile App based on device heterogeneity and OS versions.

—**As Individual Tester:** MobileMonkey focuses on blackbox testing and does not require access to source code. As a result, any testing team member can provide values to the configuration file and start testing.

—**As Developer:** The low memory footprint ensures MobileMonkey being used in development machine. A developer can run his instrumentation test and use MobileMonkey at the same time to test the context related stress handling of the app.

In essence, the holistic view on proposed MobileMonkey underscores that it will facilitate app developers with stress testing on the basis of contextual information, static analysis and myriad of execution traces, crash logs and replicable sequence of events that might be insightful for the developers.

4. EMPIRICAL STUDY AND RESULT ANALYSIS

After creating MobileMonkey, the process model from [11] has been followed to define the empirical study. At first, it has been conceptualized that the utilization of MobileMonkey should be beneficial to mobile app developers by introducing contextual scenarios that will help stress test. However, the measurement metrics had to be clearly defined. Therefore, based on the conception, the following null hypothesis and alternative hypothesis are raised:

(1) **Null Hypothesis:** Using MobileMonkey for stress testing will not create any more significant effect than industry standard tool as it will raise equal or less warnings,errors, failures or crashes in mobile apps dependent on contexts.
(2) **Alternative Hypothesis:** Using MobileMonkey stress testing will help produce better quality mobile app than industry standard tool since MobileMonkey will raise more warnings, errors, failures, or crashes in mobile apps dependent on contexts.

For the experiment, Monkey [7] was chosen as industry standard tool since it is found to be better performing than any other state-of-the-art tools or approaches [33]. At first, it was determined

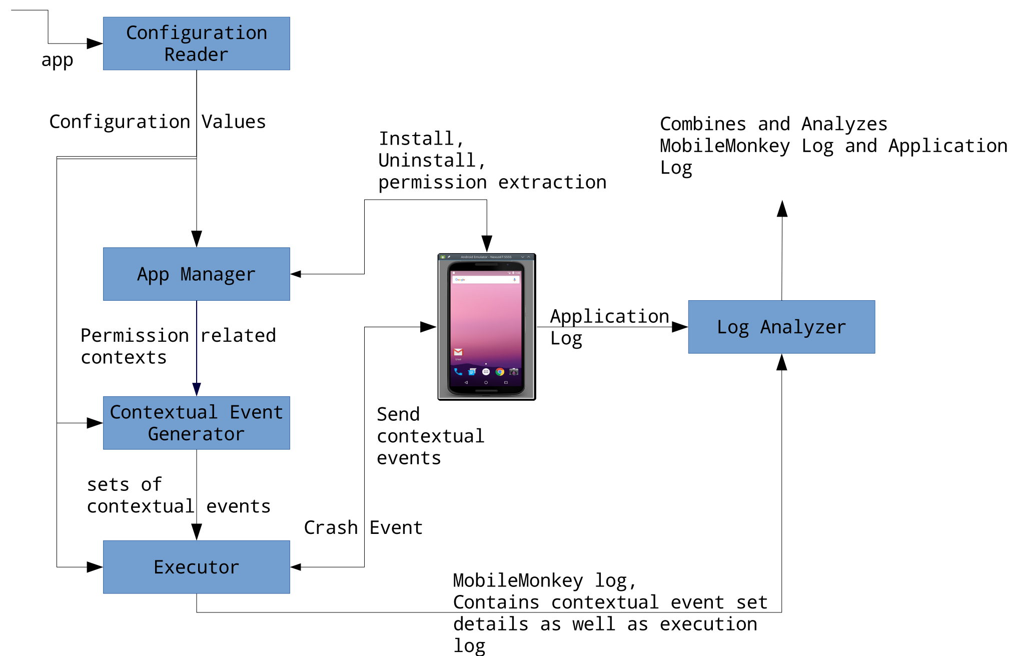

Fig. 1. MobileMonkey Flow

that there will be two types of trials; with one being treated with MobileMonkey and the other treated with Monkey. Next, it is a within-subjects experiment. Therefore, both treatments are applied on same experimental subjects, e.g. mobile apps. Third, The *Experimental objects* have to represent real world mobile apps and/or existing data set experimented on by others in academia. 20 apps from AndroTest repository [10] were chosen. 15 more apps were chosen representing real world based on the following criteria:

—is available as a released mobile app in Google Play Store

—is updated frequently, with latest update in 2017

—is downloaded more than 1 million times

—is dependent on contexts

—has a user rating of 4.0 out of 5.0 or above

Last criterion was chosen specifically to ensure that the real world apps are quality ensured. Additionally, it was determined to avoid apps that required sign-in before accessing any of its features and to focus on apps which are usable right after installation. Fourth, for the *Control Situation* the following criteria are determined:

—the duration of trial is 300 seconds

—each trial will be repeated for 3 times

—Monkey will generate 300 specific events both individually and as part of MobileMonkey. Each event will be executed with 500 milliseconds delay. While executing, Monkey takes less time as it groups up events as a single event during execution.

—Monkey will be specified to app activity only by the package specific switch (`-p`).

—Same seed will be used for Monkey and MobileMonkey

—Emulated Device with Android version Nougat (7.0) x86, 1.5GB RAM, 800MB internal storage, 2 multi-core CPUs with screen size and other specification similar to Nexus6 [16]

—To ensure each trial are independent, the emulator is hard reset after each trial

—Internet is made available for all trials

—The minimum interval is set to be 5 seconds, and maximum is 8 seconds for MobileMonkey. The Uniform interval is set to be 24 seconds for `Gsm Profile`.

For tools, the latest available binaries were used during the empirical study. For Android SDK, the version utilized was 26.0.2, the emulator was of version 26.0.3. Finally, To measure and determine whether the Null Hypothesis or the alternative Hypothesis stands, the following incidents were considered as *Response Variables*:

—**Warning:** the app does not crash, but the app functionality didn't work properly work

—**Failure:** the app does not crash, but malfunction occurred, perhaps an internal activity crashed. For example, `android.view.WindowLeaked` is an error type incident that indicates that a `dialog` was dismissed improperly.

—**Error/Crash:** the app is forced to close, or some severe error occurred, but app continued execution. For example, an error of `TextToSpeech` may take place due to absence of Text to Speech Engine in the OS.

It should be emphasized that the unique incidents were counted once in several cases. For example, if `Deeplink not found` is shown 5 times, it was counted as one incident. The message is fairly simple and shows the absence of a requested resource. Additionally, it was intervened specifically due to new permission system introduced in Android 7 right after installation of the app where user

Table 3. Comparison of MobileMonkey with Monkey [Total, Unique, Fatal] / [Warning, Error] - [T,U,F] / [W,E]

	App	**Category**	**Version**	MobileMonkey					**Monkey**				
				TW	**UW**	**TE**	**UE**	**FE**	**TW**	**UW**	**TE**	**UE**	**FE**
AndroTest Repo Apps	hu.vsza.adsdroid	Generic	1.6	4	4	9	7	1	0	0	3	3	1
	com.ringdroid	Audio	2.7.4	5	3	1	1	0	3	3	1	1	0
	com.templaro.opsiz.aka	Utility	1	13	3	0	0	0	14	3	0	0	0
	org.totschnig.myexpenses	Business	2.7.9	0	0	4	3	0	0	0	0	0	0
	cri.sanity	Utility	2.11	3	2	18	10	0	0	0	0	0	0
	org.tomdroid	Utility	0.7.5	4	3	5	5	0	5	4	2	2	0
	net.jaqpot.netcounter	Utility	0.14.1	1	1	2	2	0	1	1	0	0	0
	jp.gr.java_conf.hatalab.mnv	Productivity	0.4	2	1	0	0	0	2	1	0	0	0
	net.fercanet.LNM	Productivity	1.4	2	2	4	3	0	1	1	3	3	0
	com.bwx.bequick	Utility	1.9.9.3	17	3	4	3	0	0	0	4	3	0
	caldwell.ben.bites	Pro	1.3	5	0	0	0	0	3	1	0	0	0
	com.chmod0.manpages	Utility	1.51	210	10	9	7	0	16	5	3	3	0
	com.gluegadget.hndroid	Utility	0.2.1	5	5	1	1	0	0	0	5	5	0
	com.zoffcc.applications.aagtl	Productivity	1.0.36	35	8	7	7	0	14	6	7	7	0
	org.liberty.android.fantastischmemo	Utility	10.9.993	2	1	7	6	1	5	2	0	0	0
Real World Apps	com.bikroy	shopping	1.0.9	68	13	3	3	0	23	11	3	3	0
	com.daraz.android	shopping	2.8.2	17	11	5	5	0	4	2	0	0	0
	com.hostelworld.app	travel	5.12.0	35	11	3	3	0	7	7	1	1	0
	com.alibaba.aliexpresshd	shopping	5.3.2	36	33	23	18	0	49	30	29	18	0
	com.accuweather.android	weather	4.7.4	110	42	43	19	0	97	36	42	17	0
	com.ebay.mobile	shopping	5.11.0.12	12	5	3	3	0	20	16	0	0	0
	jp.gocro.smartnews.android	news	4.1.13	20	16	6	3	0	77	22	435	10	0
	com.apalon.weatherlive.free	weather	5.3	193	53	20	7	0	58	15	4	4	0
	com.ixigo.train.ixitrain	travel	3.5.5	26	20	0	0	0	33	19	7	5	0
	com.popularapp.thirtydayfitnesschallenge	health	1.0.25	29	7	165	90	0	1	1	98	50	0
	com.reddit.frontpage	news	2.9.1	31	14	6	3	0	24	16	4	3	0
	de.motain.iliga	news	9.5.0	565	20	6	3	0	2	2	0	0	0
	com.guardian	news	4.27.1152	56	28	33	18	0	52	26	29	15	0
	de.wetteronline.wetterapp	weather	4.4.2	37	20	23	3	0	16	13	8	3	0
	com.cricbuzz.android	sports	3.2.5	166	36	31	15	0	138	34	30	12	0

will have to explicitly allow permissions for an app when started for the first time. Based on these, the empirical study was done. The results are provided in Table 3. As shown, MobileMonkey outperforms the industry standard Monkey tool in almost all real world and AndroTest Repo Apps in terms of total number of warnings, unique warnings, total errors and unique errors. As shown in the Table 3, MobileMonkey outperforms Monkey in 24 cases in total warnings and 26 cases of total errors. In several cases, MobileMonkey is also able to fatally crash the android app being tested, which monkey could not in our experimental study.

5. CONCLUSION

In this paper, we present MobileMonkey, a contextual stress testing framework form Android App developers. The proposed framework provides a systematic analysis of bugs, crashes and execution traces on the basis of static analysis and contextual execution which will provide insightful data regarding discovering what went wrong during app development. We evaluated MobileMonkeys effectiveness in comparison with industry standard Android app stress testing tools. The results demonstrate that MobileMonkey consistently performs better than the industry standard tool for stress testing in a diverse range of scenarios with desirable properties of being resource friendly, horizontally scalable and SDK agnostic. More advanced approach will be explored for based on richer set of contexts, learning of mobility scenarios, model based system calls etc.

ACKNOWLEDGEMENT

The work is supported by Innovation Fund of fiscal year 2014-15, reference code: 3-0001-2801-5965, ICT Division, Peoples' Republic of Bangladesh.

6. REFERENCES

[1] Akamai. Q1 2017 State of the Internet - Connectivity Report — Akamai. Technical report, Akamai Technologies, 2017.

[2] Domenico Amalfitano, Anna Rita Fasolino, and Porfirio Tramontana. A GUI Crawling-Based Technique for Android Mobile Application Testing. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 252–261. IEEE, mar 2011.

[3] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, and Bryan Robbins. Testing Android Mobile Applications: Challenges, Strategies, and Approaches. *Advances in Computers*, 89:1–52, 2013.

[4] Android.com. Settings.Global — Android Developers. (https:// developer.android.com/ reference/android/provider/Settings.Global.html, accessed 28-May-2017).

[5] Android.com. Android Device Monitor — Android Studio. (https://developer.android.com/studio/profile/monitor.html, accessed 28-May-2017).

[6] Android.com. Logcat Command-line Tool — Android Studio . (https://developer.android.com/studio/command-line/logcat.html, accessed 28-May-2017).

[7] Android.com. UI/Application Exerciser Monkey — Android Studio. (https:// developer.android.com/studio/test/monkey.html, accessed 28-May-2017).

[8] Appbrain.com. Ratings on Google Play - AppBrain. (https://www.appbrain.com/stats/android-app-ratings, accessed 28-May-2017).

[9] Apple.com. App Store Review Guidelines - Apple Developer. (https://developer.apple.com/app-store/review/guidelines/#app-completeness, accessed 28-May-2017).

[10] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. Automated test input generation for android: Are we there yet? In *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*, 2016.

[11] Norman Fenton and James Bieman. *Software Metrics: A Rigorous and Practical Approach, Third Edition*. CRC Press, Inc., Boca Raton, FL, USA, 3rd edition, 2014.

[12] Gameapp.gov.bd. Skill Development for Mobile Game and Application. (http://gameapp.gov.bd/, accessed 28-May-2017).

[13] Googleblog.com. Android Developers Blog: I/O 2017: Everything new in the Google Play Console. (https://android-developers.googleblog.com/2017/05/whats-new-in-google-play-at-io-2017.html, accessed 28-May-2017).

[14] Google.com. Firebase. (https://firebase.google.com/pricing/, accessed 28-May-2017).

[15] Google.com. Firebase Test Lab for Android — Firebase. (https://firebase.google.com/docs/test-lab/, accessed 28-May-2017).

[16] Google.com. Nexus 6P - Google. (https://www.google.com/nexus/6p/, accessed 28-May-2017).

[17] imedicalapps.com. iMedicalApps - Reviews of Medical apps & Healthcare Technology. (https://www.imedicalapps.com/, accessed 28-May-2017).

[18] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E. Hassan. What Do Mobile App Users Complain About? *IEEE Software*, 32(3):70–77, may 2015.

[19] Chieh-jan Mike Liang, Nicholas D Lane, Niels Brouwers, Li Zhang, Börje F. Karlsson, Hao Liu, Yan Liu, Jun Tang, Xiang Shan, Ranveer Chandra, and Feng Zhao. Caiipa: Automated Large-scale Mobile App Testing through Contextual Fuzzing. *MobiCom*, pages 519–530, 2014.

[20] CJM Liang, ND Lane, Niels Brouwers, and Li Zhang. Context Virtualizer: A Cloud Service for Automated Large-scale Mobile App Testing under Real-World Conditions. *Msr-Waypoint.Com*, MSR-TR-201, 2013.

[21] Aravind Machiry, Rohan Tahiliani, and Mayur Naik. Dynodroid: an input generation system for Android apps. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*, page 224, 2013.

[22] Kevin Moran, Mario Linares-Vasquez, Carlos Bernal-Cardenas, Christopher Vendome, and Denys Poshyvanyk. Automatically Discovering, Reporting and Reproducing Android Application Crashes. In *Proceedings - 2016 IEEE International Conference on Software Testing, Verification and Validation, ICST 2016*, pages 33–44, 2016.

[23] Henry Muccini, Antonio Di Francesco, and Patrizio Esposito. Software testing of mobile applications: Challenges and future research directions. In *Proceedings of the 7th International Workshop on Automation of Software Test*, AST '12, pages 29–35, Piscataway, NJ, USA, 2012. IEEE Press.

[24] Rajesh Palit, Renuka Arya, Kshirasagar Naik, and Ajit Singh. Selection and execution of user level test cases for energy cost evaluation of smartphones. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST '11, pages 84–90, New York, NY, USA, 2011. ACM.

[25] Fabio Palomba, Mario Linares-Vasquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. User reviews matter! Tracking crowdsourced reviews to support evolution of successful apps. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 291–300. IEEE, sep 2015.

[26] Lenin Ravindranath, Suman Nath, Jitendra Padhye, and Hari Balakrishnan. Automatic and scalable fault detection for mobile applications. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services - MobiSys '14*, pages 190–203, New York, New York, USA, 2014. ACM Press.

[27] Raimondas Sasnauskas and John Regehr. Intent fuzzer: crafting intents of death. In *Proceedings of the 2014 Joint International Workshop on Dynamic Analysis (WODA) and Software and System Performance Testing, Debugging, and Analytics (PERTEA) - WODA+PERTEA 2014*, pages 1–5, New York, New York, USA, 2014. ACM Press.

[28] squareup.com. Credit Card Processing - Accept Credit Cards Anywhere — Square. (https://squareup.com/, accessed 28-May-2017).

[29] Techcrunch.com. Users Have Low Tolerance For Buggy Apps Only 16% Will Try A Failing App More Than Twice — TechCrunch. (https://techcrunch.com/2013/03/12/users-have-low-tolerance-for-buggy-apps-only-16-will-try-a-failing-app-more-than-twice, accessed 28-May-2017).

[30] tourism.gov.in. Mobile Application For Tourist on Google play — Ministry of Tourism. (http://tourism.gov.in/mobile-application-tourist-google-play, accessed 28-May-2017).

[31] Anthony I Wasserman and Fosser. Software Engineering Issues for Mobile Application Development. In *FoSER '10 Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 397–400, 2010.

[32] Xamarin.com. Mobile App Testing On Hundreds Of Devices - Xamarin Test Cloud. (https://www.xamarin.com/test-cloud, accessed 28-May-2017).

[33] Xia Zeng, Dengfeng Li, Wujie Zheng, Fan Xia, Yuetang Deng, Wing Lam, Wei Yang, and Tao Xie. Automated Test Input Generation for Android : Are We Really There Yet in an Industrial Case ? pages 3–8, 2015.