Managing App Testing Device Clouds: Issues and Opportunities

Mattia Fazzini
University of Minnesota
Minneapolis, MN, USA
mfazzini@umn.edu

Alessandro Orso
Georgia Institute of Technology
Atlanta, GA, USA
orso@cc.gatech.edu

ABSTRACT

Because creating and maintaining an in-house test lab is expensive and time-consuming, companies and app developers often use device clouds to test their apps. Because quality-assurance activities depend on such device clouds, it is important to understand possible issues related to their use. To this end, in this paper we present a preliminary study that investigates issues and highlights research opportunities in the context of managing and maintaining device clouds. In the study, we analyzed over 12 million test executions on 110 devices. We found that the management software of the cloud infrastructure we considered affected some test executions, and almost all the cloud devices had at least one security-related issue.

ACM Reference Format:
Mattia Fazzini and Alessandro Orso. 2020. Managing App Testing Device Clouds: Issues and Opportunities. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), September 21–25, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3324884.3418909

1 INTRODUCTION

Thorough in-house testing of Android apps has been particularly challenging for companies and developers, due to the fragmentation of the Android ecosystem [6, 11, 12, 14, 16, 18, 19, 21, 26, 34–36, 40]. Because of the extremely large number of devices and versions of the Android operating system running in the field, companies and developers are required to check that their apps behave as expected on a large set of devices. However, creating and maintaining an in-house test lab with a large set of devices is expensive, difficult, and ultimately impractical [7].

To support companies and developers, many organizations [1, 9, 13, 25, 29, 30] built publicly accessible device clouds that developers can use to thoroughly test their apps. Specifically, developers can use these cloud devices to run tests on multiple devices, monitor test execution progress, and retrieve text execution artifacts.

Understanding possible issues with device-cloud infrastructure is extremely important, as these issues might affect the quality assurance processes that rely on such infrastructure. In this spirit, we performed a preliminary study that aims to investigate issues, as well as highlight research opportunities, in the context of managing and maintaining cloud-based app-testing infrastructure. In the study, we ran the set of tests from Google's Android Compatibility Test Suite [8] on the devices in the AWS Device Farm [1] and used the tests to identify issues in this device cloud. After analyzing more than 12 million test executions on 110 devices, we found that (1) the cloud management software used to run and update devices can interfere with test executions and (2) devices tend to have security-related issues that could be exploited by attackers.

Table 1: Android versions, devices, and tests in our study.

Android Version	*# Devices*	*CTS Version Tag*	*# Modules*	*# Tests*
Jelly Bean 4.2	2	4.2.2_r1	50	17268
Jelly Bean 4.3	5	4.3_r2.2-cts	57	18013
KitKat 4.4	38	cts-4.4_r4	73	24039
Lollipop 5.0	11	cts-5.0_r9	93	33929
Lollipop 5.1	6	cts-5.1_r28	96	34207
Marshmallow 6.0	15	cts-6.0_r32	114	37613
Nougat 7.0	7	cts-7.0_r32	124	43494
Nougat 7.1	2	cts-7.1_r28	133	43598
Oreo 8.0.0	11	cts-8.0_r20	195	59459
Oreo 8.1.0	4	cts-8.1_r16	210	60383
Pie 9	9	cts-9.0_r9	289	84182

2 METHODOLOGY

As mentioned above, we investigated possible issues with device clouds by executing the set of tests in Google's Compatibility Test Suite (CTS) [8] on devices hosted in the AWS Device Farm (DF) [1].

Google's CTS provides a mechanism for checking whether a device's Android operating system (OS) exhibits standard behavior; devices that pass all CTS tests are considered *Android compatible*. This helps ensure that app developers can rely on a consistent execution environment despite the various OS customizations performed by device vendors [6, 34, 35].

We selected DF as our cloud testing environment because (i) is a popular environment for mobile app testing [13, 29, 30], (ii) offers access to the machine driving the test suite execution (which is necessary to run the CTS tests), and (iii) has a wide variety of devices running different versions of the Android OS.

In our study, we considered devices certified as Android compatible and checked whether they passed all the CTS tests also in the cloud testing environment. Because these devices passed all CTS tests when the tests were run locally, test failures on the cloud devices would likely indicate issues in the cloud infrastructure.

Table 1 shows relevant information for the devices we considered and their corresponding Android OS versions. For each version of the Android OS (*Android Version*), the table shows the number of devices running that version (*# Devices*), the CTS version used (*CTS Version Tag*),[1] the number of test modules (*# Modules*), where a test module contains tests that exercises a specific functionality,[2] and the overall number of tests.

[1]For each OS version, we used the latest release of the CTS available for that version.
[2]We disregarded some test modules because they either required a specific device configuration that we could not set programmatically, or their execution time exceeded the time limit enforced by the DF.

ASE '20, September 21–25, 2020, Virtual Event, Australia

ACM ISBN 978-1-4503-6768-4/20/09…$15.00
https://doi.org/10.1145/3324884.3418909

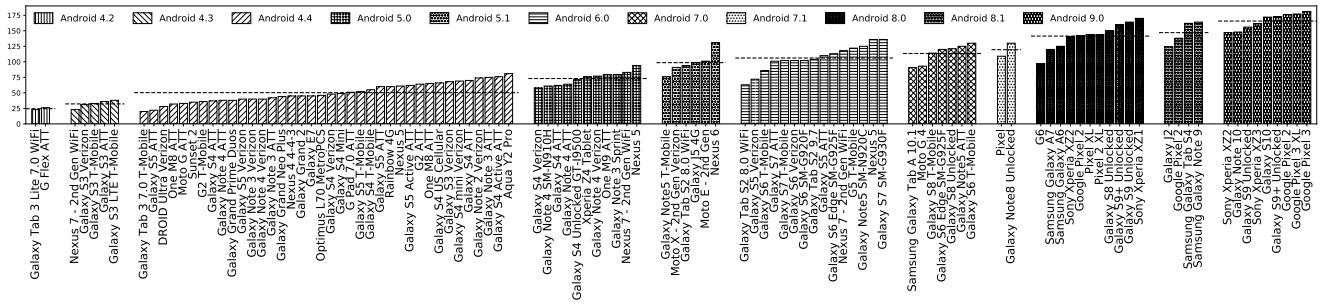

Figure 1: Test failures for the different devices considered.

Note that, to account for test flakiness [5, 22, 33], we executed each test three times and considered a test as passing if it passed at least once. This configuration led to the execution of over 12 million tests, which took 136 days of machine time to complete.

3 RESULTS AND ANALYSIS

Figure 1 summarizes the number of failures triggered by the tests. The x-axis lists the devices considered (make, name, and carrier), whereas the y-axis reports the number of failures for those devices. Devices are grouped by their OS version, using different fill patterns and with the most recent versions last. Overall, although most tests passed, we observed 9,778 failures, and all devices experienced at least one failure.

As a first investigation into the causes of these failures, we identified tests that failed on all devices on which they ran, as these tests should reveal general issues with the cloud infrastructure. This resulted in the identification of 58 tests from 15 different modules. Intuitively, We then met with engineers from the DF team to discuss the results, and they confirmed that the failures for 27 of these tests were indeed caused by the cloud infrastructure, and in particular by the fact that the software used to manage the cloud devices inhibits certain types of screen animations. Therefore, when using these cloud devices, app developers might experience test failures that are not caused by faults in their apps and are thus false positives. For the remaining 21 failures, the DF engineers were still collecting the data produced in our experiments, so we do not yet have a final confirmation. In future work, we plan to extend this part of our study by analyzing failures on a per-vendor and per-version basis, as cloud-related failures might also be vendor- or version-specific. We will also investigate how light-weight monitoring of test executions could help identify situations in which test results are affected by software external to the apps being testes, so as to suitably inform developers.

We also analyzed our results to detect whether cloud devices have security-related issues that can be exploited. When new bugs, and in particular security bugs (i.e., vulnerabilities) are found in the Android OS, new tests are added to the CTS to prevent new devices from being released with the same bugs. Devices that are already running in the field should receive updates to fix these bugs, and so should devices that are running within a cloud infrastructure. We therefore investigated whether cloud devices are affected by known security bugs that attackers could exploit (and that could make test results unreliable). First, we identified security-related (failing) tests by manually analyzing package names, test names, and stack traces of failing tests. Across all devices and versions, we found 1,307 security-related test failures (13% of all failures) caused by 153 different tests. All devices but one had at least one security-related test failure. For example, test `testStagefright_cve_2016_2507` revealed that seven of the considered devices are affected the vulnerability described in CVE-2016-2507 [4] (an arbitrary code execution vulnerability). This result motivates the investigation and use of techniques for performing run-time security monitoring of cloud devices and for frequently and quickly delivering updates to the devices when problems are detected.

4 RELATED WORK

Our work mostly relates to cloud-based testing of mobile apps [2, 3, 10, 23, 24, 27, 28, 31, 32, 37–39] and compatibility analysis of mobile software [6, 12, 15, 17, 19, 20, 34, 35]. The former research area focuses on designing and implementing cloud-based testing infrastructure [24, 28, 31, 32] and on improving the process of testing mobile apps in the cloud [10, 23, 27, 39]. The latter research area focuses on identifying compatibility issues between different devices [6, 15, 17, 34, 35] and between different releases of Android [12, 19]. Although this previous work is related to ours, to the best of our knowledge, this is the first study that systematically studies issues in a real-world cloud testing infrastructure.

5 CONCLUSION

In this paper, we presented a preliminary study that identified issues and research opportunities in the context of managing and maintaining a cloud-based app-testing infrastructure. Our results show that the cloud infrastructure can interfere with the tests run on cloud devices and cause spurious failures. They also show that cloud devices suffer from vulnerabilities that could be exploited by malicious users. In the future, we plan to expand our study by creating a detailed taxonomy of the failures. We also plan to perform a per-vendor and per-version analysis of test failures. Our study results also motivate research in the areas of automatic generation of compatibility-based tests and light-weight monitoring of cloud-based test executions.

ACKNOWLEDGMENTS

This work was partially supported by gifts from Amazon and Facebook, and by NSF grant CCF-1563991.

REFERENCES

[1] Amazon. 2020. *AWS Device Farm.* Retrieved August 31, 2020 from https://aws.amazon.com/device-farm

[2] Antonia Bertolino, Guglielmo De Angelis, Micael Gallego, Boni García, Francisco Gortázar, Francesca Lonetti, and Eda Marchetti. 2019. A Systematic Review on Cloud Testing. *ACM Comput. Surv.* (2019).

[3] L. Cheng, J. Chang, Z. Yang, and C. Wang. 2016. GUICat: GUI testing as a service. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)* (Singapore, Singapore). IEEE, 858–863.

[4] CVE. 2016. *CVE-2016-2507.* Retrieved August 31, 2020 from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-2507

[5] Zhen Dong, Abhishek Tiwari, Xiao Liang Yu, and Abhik Roychoudhury. 2020. Concurrency-related Flaky Test Detection in Android apps. arXiv:2005.10762 [cs.SE]

[6] Mattia Fazzini and Alessandro Orso. 2017. Automated Cross-Platform Inconsistency Detection for Mobile Apps. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering* (Urbana-Champaign, IL, USA). IEEE, 308–318.

[7] Daniel Giordano. 2017. *The Average Cost Of A Device Lab.* Retrieved August 31, 2020 from https://crossbrowsertesting.com/blog/browsers/average-cost-device-lab

[8] Google. 2020. *Compatibility Test Suite.* Retrieved August 31, 2020 from https://source.android.com/compatibility/cts

[9] Google. 2020. *Firebase Test Lab.* Retrieved August 31, 2020 from https://firebase.google.com/docs/test-lab

[10] P. Graubner, L. Baumgärtner, P. Heckmann, M. Müller, and B. Freisleben. 2015. Dynalize: Dynamic Analysis of Mobile Apps in a Platform-as-a-Service Cloud. In *2015 IEEE 8th International Conference on Cloud Computing* (New York, NY, USA). IEEE, 925–932.

[11] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia. 2012. Understanding Android Fragmentation with Topic Analysis of Vendor-Specific Bugs. In *2012 19th Working Conference on Reverse Engineering* (Kingston,, Canada). IEEE, 83–92.

[12] Dongjie He, Lian Li, Lei Wang, Hengjie Zheng, Guangwei Li, and Jingling Xue. 2018. Understanding and Detecting Evolution-induced Compatibility Issues in Android Apps. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, Montpellier, France, 167–177.

[13] Software Testing Help. 2020. *Best Cloud-Based Mobile App Testing Service Providers In 2020.* Retrieved August 31, 2020 from https://www.softwaretestinghelp.com/cloud-mobile-testing-services

[14] Andreas Holzinger, Peter Treitler, and Wolfgang Slany. 2012. Making apps useable on multiple different mobile platforms: On interoperability for business application development on smartphones. In *International Conference on Availability, Reliability, and Security* (Prague, Czech Republic). IEEE, 176–189.

[15] J. Huang. 2014. AppACTS: Mobile App Automated Compatibility Testing Service. In *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering* (Oxford, UK). IEEE, 85–90.

[16] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. 2013. Real Challenges in Mobile app Development. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Baltimore, MD, USA). IEEE, 15–24.

[17] Taeyeon Ki, Chang Min Park, Karthik Dantu, Steven Y. Ko, and Lukasz Ziarek. 2019. Mimic: UI Compatibility Testing System for Android Apps. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada). IEEE Press, 246–256.

[18] Huoran Li, Xuan Lu, Xuanzhe Liu, Tao Xie, Kaigui Bian, Felix Xiaozhu Lin, Qiaozhu Mei, and Feng Feng. 2015. Characterizing smartphone usage patterns from millions of Android users. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference* (Tokyo, Japan). ACM, 459–472.

[19] Li Li, Tegawendé F. Bissyandé, Haoyu Wang, and Jacques Klein. 2018. CiD: Automating the Detection of API-related Compatibility Issues in Android Apps. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, Amsterdam, Netherlands, 153–163.

[20] C. Liu, W. Chen, and S. Chen. 2016. A Concurrent Approach for Improving the Efficiency of Android CTS Testing. In *2016 International Computer Symposium (ICS)* (Chiayi, Taiwan). IEEE, 611–615.

[21] Yepang Liu, Chang Xu, and Shing-Chi Cheung. 2014. Characterizing and detecting performance bugs for smartphone applications. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India). ACM, 1013–1024.

[22] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. 2014. An Empirical Analysis of Flaky Tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China). ACM, 643–653.

[23] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek, and A. Stavrou. 2012. A whitebox approach for automated security testing of Android applications on the cloud. In *2012 7th International Workshop on Automation of Software Test (AST)* (Zurich, Switzerland). IEEE, 22–28.

[24] A. Malini, N. Venkatesh, K. Sundarakantham, and S. Mercyshalinie. 2014. Mobile application testing on smart devices using MTAAS framework in cloud. In *International Conference on Computing and Communication Technologies* (Hyderabad, India). IEEE, 1–5.

[25] Microsoft. 2020. *Visual Studio App Center.* Retrieved August 31, 2020 from https://appcenter.ms

[26] Abhinav Pathak, Y Charlie Hu, and Ming Zhang. 2011. 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* (Cambridge, MA, USA). ACM, 1–6.

[27] C. M. Prathibhan, A. Malini, N. Venkatesh, and K. Sundarakantham. 2014. An automated testing framework for testing Android mobile applications in the cloud. In *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies* (Ramanathapuram, India). IEEE, 1216–1219.

[28] Isabel K. Villanes Rojas, Silvia Meireles, and Arilo Claudio Dias-Neto. 2016. Cloud-Based Mobile App Testing Framework: Architecture, Implementation and Execution. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing* (Maringa, Parana, Brazil). ACM, 1–10.

[29] Snigdha. 2020. *Top Device Clouds for Mobile App Testing.* Retrieved August 31, 2020 from https://www.appypie.com/top-device-clouds-for-mobile-app-testing

[30] SYSTANGO. 2020. *Everything You Need to know about Cloud-Based Device Testing Providers.* Retrieved August 31, 2020 from https://medium.com/@systango/everything-you-need-to-know-about-cloud-based-device-testing-providers-3090fe465a58

[31] Chuanqi Tao and Jerry Gao. 2017. On building a cloud-based mobile testing infrastructure service system. *Journal of systems and software* 124 (2017), 39–55.

[32] C. Tao, J. Gao, and B. Li. 2015. Cloud-Based Infrastructure for Mobile Testing as a Service. In *2015 Third International Conference on Advanced Cloud and Big Data* (Yangzhou, China). IEEE, 133–140.

[33] S. Thorve, C. Sreshtha, and N. Meng. 2018. An Empirical Study of Flaky Tests in Android Apps. In *2018 IEEE International Conference on Software Maintenance and Evolution* (Madrid, Spain). IEEE, 534–538.

[34] Lili Wei, Yepang Liu, and Shing-Chi Cheung. 2016. Taming Android Fragmentation: Characterizing and Detecting Compatibility Issues for Android Apps. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore). ACM, 226–237.

[35] Lili Wei, Yepang Liu, and Shing-Chi Cheung. 2019. Pivot: Learning API-Device Correlations to Facilitate Android Compatibility Issue Detection. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada). IEEE, 878–888.

[36] Lei Wu, Michael Grace, Yajin Zhou, Chiachih Wu, and Xuxian Jiang. 2013. The impact of vendor customizations on android security. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (Berlin, Germany). ACM, 623–634.

[37] M. G. Xavier, K. J. Matteussi, G. R. França, W. P. Pereira, and C. A. F. De Rose. 2017. Mobile Application Testing on Clouds: Challenges, Opportunities and Architectural Elements. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)* (St. Petersburg, Russia). IEEE, 181–185.

[38] Samer Zein, Norsaremah Salleh, and John Grundy. 2016. A Systematic Mapping Study of Mobile Application Testing Techniques. *J. Syst. Softw.* (2016), 334–356.

[39] S. Zhang and B. Pi. 2015. Mobile Functional Test on TaaS Environment. In *2015 IEEE Symposium on Service-Oriented System Engineering* (San Francisco, CA, USA). IEEE, 315–320.

[40] Xiaoyong Zhou, Yeonjoon Lee, Nan Zhang, Muhammad Naveed, and XiaoFeng Wang. 2014. The peril of fragmentation: Security hazards in android device driver customizations. In *2014 IEEE Symposium on Security and Privacy* (San Jose, CA, USA). IEEE, 409–423.