Hao Lin *Tsinghua University, Beijing, China* **Jiaxing Qiu, Hongyi Wang** *Tsinghua University and ByteDance Inc., Beijing, China*
Zhenhua Li *Tsinghua University, Beijing, China* **Liangyi Gong** *CNIC of CAS, Beijing, China*
Di Gao, Yunhao Liu *Tsinghua University, Beijing, China* **Feng Qian** *University of Southern California, CA, USA*
Zhao Zhang, Ping Yang *ByteDance Inc., Beijing, China* **Tianyin Xu** *University of Illinois at Urbana-Champaign, IL, USA*

Editor: Steve Ko

TAKE THE BLUE PILL: PURSUING MOBILE APP TESTING FIDELITY, EFFICIENCY, AND ACCESSIBILITY WITH VIRTUAL DEVICE FARMS

Illustration, istockphoto.com

For more than a decade, researchers have been extensively exploring mobile app testing on *virtual devices* [1-9], which are software-emulated mobile devices running on commodity servers, in a similar vein as virtual machines (VM) in the cloud. Building on server virtualization, virtual devices naturally inherit the benefits of VM, such as scalability, elasticity, and cost efficiency. Moreover, virtualization enables useful features not offered by physical devices, such as service instrumentation [3], whole-system snapshot [9], and memory introspection [1,2], atop which a series of advanced testing and debugging techniques are developed.

In contrast to the wide adoption of virtual devices in academic research, it remains controversial to rely on virtual devices for mobile app testing in practice, especially in industrial settings. The main concern comes from the inherent difficulties of high-fidelity device emulation, especially in an open mobile ecosystem, such as Android where hundreds of new phone models are released every year with heterogeneous hardware and highly customized software (as exemplified in Figure 1). Discrepancies between virtual devices and physical devices may lead to both escapes of bugs and false alarms. For mobile apps with a global user base, even seemingly small-numbered discrepancies could have magnified impacts.

Consequently, major mobile app companies are instead spending millions of dollars to build and operate *physical device farms*, a dedicated infrastructure that hosts diverse phone models for comprehensive mobile app testing (Figure 2). For example, *Douyin* [10], a mobile app with 842 million users, is tested on a large-scale device farm before its version releases, which hosts a total of 5,918 different device models as of January 2022. The device farm costs more than 1M US dollars to build, and over 0.6M dollars per year for device upgrades and replacements, not including the yearly salaries of the maintenance team, carrier plan expenses of cellular-capable devices, maintenance cost of Wi-Fi networks, power usage, etc.

To reduce the operation cost, we study the essential problems of using virtual device farms for large-scale mobile app testing in industrial settings. Our goal is to (1) quantitatively understand the fidelity of virtual devices, and (2) explore how to better utilize virtual device farms to drastically improve the efficiency and accessibility of large-scale mobile app testing.

From January 1 to March 31, 2022, we conducted a comparative study of the real-world test results of Douyin and nine other global-scale mobile apps on a massive commercial testing infrastructure that deploys a physical device farm and its virtualized counterpart. We show that high testing fidelity can be achieved by sensible design and implementation of virtual device farms. Most importantly, we reveal major discrepancies in non-standard, uncoordinated, and occasionally defective

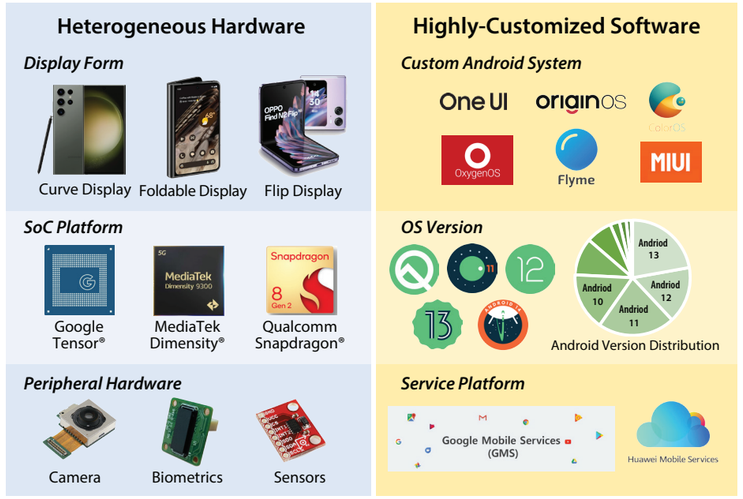


FIGURE 1. Hardware and software dynamics of the Android ecosystem.

vendor-specific services, drivers, and defense mechanisms, rather than hardware heterogeneity and system customizations. With that, we improve and use virtual device farms to build a continuous mobile app testing pipeline, which substantially improves the testing efficiency and accessibility in production. To benefit the community, we also publish the data and analysis code involved in this study at https://github.com/android-emulation-testing/emu-fidelity-ae.

BUILDING THE VIRTUAL DEVICE FARM

Aiming to build a digital twin of the physical farm (that mimics the physical devices as much as possible) and understand the fidelity limitations of existing virtual devices, we make several important design decisions in terms of hardware, mobile OS and mobile app for our virtual device farm, based on state-of-the-art mobile emulation solutions. The overall architecture of our virtual devices is shown in Figure 3.

For the hardware infrastructure, we choose ARM servers for hosting virtual devices instead of more common x86 servers. The rationales are twofold. First, with the recent development of the ARM server market, an ARM server is ~20% cheaper than an x86 server with the same number of CPU cores and memory/storage capacity. Second, the affinity between ARM servers and Android phones (all of which employ ARM SoCs) avoids the need for dynamic binary translation (DBT) from the ARM instructions of Android apps to the x86 instructions, particularly given that existing DBT support (Intel Houdini and Google libndk) for Android is neither complete (e.g., no DBT tool for Android 10) nor reliable (e.g., frequent crashes in certain apps' native libraries).

For the mobile OS running inside virtual devices (i.e., guest OS), state-of-the-art mobile emulation techniques [11,12] usually use Framework-level hooking to achieve emulation of mobile hardware, such as Wi-Fi, GPU, Bluetooth, modem, etc. Instead, we leverage virtio devices available in Android's Linux kernel for the emulation of high-throughput I/O devices (e.g., storage and GPU); for the other hardware components (e.g., various sensors), we provide pure software emulation at the HAL. In this way, we avoid Framework-level modifications of the guest OS to preserve as much source-level fidelity as possible.

To match vendors' app-related customizations, we also install the vendor-specific app service platforms, i.e., a collection of installable vendor apps and services, such as GMS (Google Mobile Services) and HMS (Huawei Mobile Services), on each

FIGURE 2. A physical device farm for comprehensive mobile app testing.

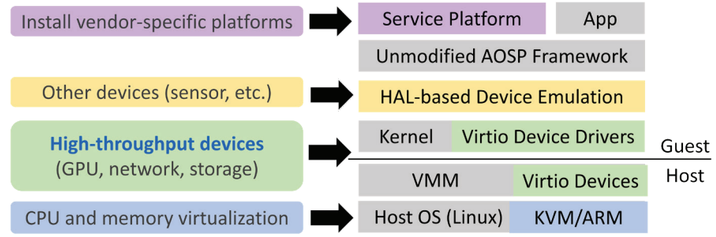


FIGURE 3. Architecture of our virtual devices.

TABLE 1. Mobile apps used in this study

App	Functionality	# Users	# Releases	Test Time
Douyin	Video streaming shopping, social media, map, education, etc.	842M	12	72 hours
Douyin Lite	Video streaming, communication, travel, photography, etc.	210M	12	72 hours
Xigua Video	Video streaming, payment, shopping, 3D gaming, etc.	180M	12	72 hours
Toutiao	News feed, shopping, web browsing, 3D gaming etc.	530M	12	72 hours
Toutiao Lite	News feed, video streaming, security checking, payment, etc.	130M	12	72 hours
Lark	Communication, email, video conference, cloud storage, etc.	9.4M	5	30 hours
Helo	Social media, video streaming, communication, etc.	50M	12	72 hours
Fizzo Novel	E-book, shopping, 3D gaming, social media, etc.	10M	5	30 hours
Xingfu Li	E-commerce, video streaming, finance, communication, etc.	7.5M	12	72 hours
Resso Music	Music streaming, communication, social media, etc.	40M	9	54 hours

virtual device. In contrast, we do not port other system-level components (system services and drivers) given their intricate dependencies on proprietary software and hardware. For example, Qualcomm drivers rely on the MSM Android Kernel Subsystems, and follow their own hardware specifications to implement register I/O and MMIO, which are proprietary and hard to emulate [13].

WHAT LEAD TO DISCREPANCIES?

We collected the test results of Douyin and nine other global-scale popular mobile apps (as detailed in Table 1) between January 1 and March 31 in 2022. For each version release of an app, we run automated tests using state-of-the-art model-based UI testing tools [14] on both the physical and virtual device farms.

Our study captures a total of 390,286 test failure events on physical and virtual devices, with 873 root causes. From the large dataset, we have several important findings regarding the actual fidelity of virtual devices in practice, and the causes of discrepancies.

Overall Fidelity. First and foremost, we answer the fidelity concern by showing that the testing fidelity of virtual devices is surprisingly good. Among all the failure events captured on the physical devices, the vast majority (92.4%) of them are also captured on the corresponding virtual devices. Meanwhile, the false positives are low – only 1.8% of failure events on virtual devices do not manifest on the physical devices. Figure 4 shows the precision and recall per app. The fidelity characteristics are similar across the apps. This indicates that with sensible design, implementation, and configuration, virtual device farms can achieve high-fidelity mobile app testing for diverse device models.

BUILDING ON SERVER VIRTUALIZATION, VIRTUAL DEVICES NATURALLY INHERIT THE BENEFITS OF VM, SUCH AS SCALABILITY, ELASTICITY, AND COST EFFICIENCY

Hardware-Level Discrepancies. At the hardware-level, counterintuitively, we show that the device emulator's lack of support for vendor-specific hardware is not a major root cause of discrepancies. This is because Android HAL provides no abstractions for vendor-specific hardware, which is thus rarely accessed by mobile apps except vendors' own apps. On the contrary,

defective drivers of common hardware on physical devices contribute to 27.6% of the discrepancies. For example, a buggy null-termination of C strings in the video codec module of Meizu's systems causes the second most (9.1%) false positives. Similarly, a bug in the GPU drivers of certain old MediaTek SoCs incurs 8.9% of false negatives.

Software-Level Discrepancies. At the software level, we find that vendor-specific Android framework customizations incur few discrepancies either. The compatibility is largely attributed to specifications enforced by Android CTS (Compatibility Test Suite) and VTS (Vendor Test Suite), which are unit-level tests to ensure functional consistency of standard Android components after customizations. However, CTS and VTS do not check interfaces between stakeholders. Consequently, add-on system services often break specifications of other stakeholders. For example, an implicit long-to-int type conversion causing integer overflow during Android's initialization of vendor-specific system services causes the most false negatives (14.9%).

Ecosystem Discrepancies. Finally, we find that there are considerable discrepancies in the occurrence frequency of certain failures among different regional mobile app ecosystems. For example, due to a lack of well-regulated app stores like Google Play, mobile users of certain regions (e.g., China) are more prone to malicious apps. In reaction, many phone vendors in those regions deploy very aggressive defense mechanisms (e.g., force killing of apps upon long background activities) to limit app behaviors; however, such mechanisms cause side effects on regular apps, such as resource leaks and data corruption. Tests on related devices manifest up to 1,025× more frequent occurrences of certain failures than on other devices.

Improving Fidelity. To improve the virtual device fidelity, we did not limit ourselves to pure technical solutions, but also coordinate stakeholders in the mobile industry. At the virtual device side where we can easily apply patches, we align the implementations like graphics format interpretations and defense mechanisms. For proprietary vendor components of physical devices, we find that vendors are often not well motivated to fix seemingly app-specific issues. To convince them to apply our fixes, we develop a dynamic binary patching technique for quickly deploying our proposed fixes at runtime (without modifying sources). The fixes then serve as a proof of causality for motivating untrusting stakeholders. As a result of these efforts, the recall of virtual devices increases from 92.4% to 94.7%, and the precision grows from 98.2% to 99.1%.

VIRTUAL DEVICES FOR CONTINUOUS TESTING

With the deep understanding of testing fidelity, we take a step further to reshape the mobile app testing infrastructure of Douyin. Previously, all the tests of Douyin before its version releases were directly conducted on the physical device farm. The results were high operation cost and reduced device lifetime.

With high-fidelity virtual devices, we developed a continuous mobile app testing pipeline to enable CI/CD, as illustrated in Figure 5. The idea is to combine the efficiency of virtual devices and the safety of physical devices – virtual devices are used to continuously test code and configuration changes to detect most functional bugs during development cycles, while physical devices are used only upon app releases as the last-level defense to capture escaped bugs. Since our continuous testing has captured most app-level bugs (which account for 93% of all bugs), tests on physical devices are much less frequently interrupted by failures.

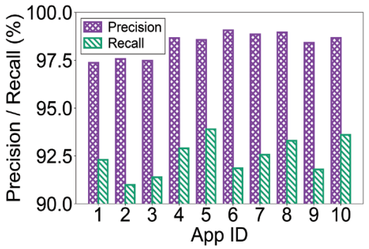
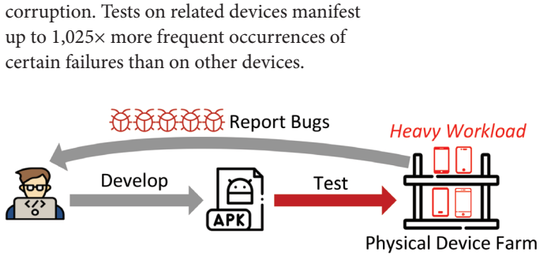


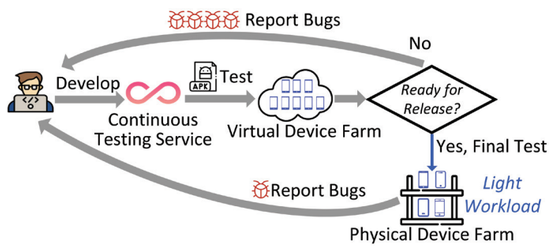
FIGURE 4. Precision and recall of the test results on virtual devices, relative to those on physical devices.

We deployed and analyzed the continuous testing pipeline in production with the ten studied apps from January 1 to February 28 in 2023. The new pipeline accelerates the end-to-end app development workflow by around 40%. Also, the device lifespan is lengthened by 1.5× on average due to reduced workload, and maintenance efforts are greatly reduced. The result is ~3× reduction in the total operation cost.

We recently also started to make our virtual devices an accessible service upon request, targeting app developers who cannot afford numerous physical devices. Preliminary feedback indicates that compared to developers' current testing practices using a small number of physical devices, our service helps detect 3–10× more bugs. Also, the feedback shows that our major conclusions can generalize to a broader range of apps.



(a) Traditional app testing on physical farms.



(b) Continuous app testing with virtual device farms.

FIGURE 5. Comparisons between traditional and continuous mobile app testing workflows.

CONCLUSION

Our work shows that virtual devices can provide immense utilities for effective continuous app testing at scale, despite their inherent difficulties of high-fidelity emulation across diverse mobile systems and hardware devices. Our experiences tell that, with careful design, implementation, and configuration, the essential fidelity gap can be effectively closed to achieve high-fidelity app testing. Although it is still hard to rule out all possible discrepancies, high-fidelity virtual device farms can substantially improve developer productivity as well as the sustainability of physical device farms. Furthermore, the elasticity and accessibility of virtual devices could enable new testing infrastructures as services for mobile apps, benefiting app developers who cannot afford large physical device farms. ■

Funding Acknowledgement

This work is supported in part by National Key R&D Program of China under grant 2022YFB4500703, National Natural Science Foundation of China under grants 62332012, 61902211 and 62202266, Microsoft Research Asia, and the Ant Group. Tianyin Xu is supported in part by NSF CNS-1956007 and CNS-2145295.

Hao Lin is a PhD student in the School of Software at Tsinghua University. His research interests are in high-performance and reliable mobile systems.

Jiaxing Qiu is a PhD student in the School of Software at Tsinghua University. His research interests are in mobile systems and virtualization.

Hongyi Wang is a PhD student in the School of Software at Tsinghua University. Her research interests are in mobile systems and data mining.

Zhenhua Li is a tenured associate professor in the School of Software at Tsinghua University. His research areas cover network measurement, mobile networking/emulation, and cloud computing/storage.

Liangyi Gong is a senior engineer at the Computer Network Information Center (CNIC), Chinese Academy of Sciences. His research interests are in the broad areas of mobile networking, system and security.

Di Gao is a PhD student in the School of Software at Tsinghua University. His research interests are in operating system kernel and virtualization.

Yunhao Liu is a full professor and the Dean of Global Innovation Exchange (GIX), Tsinghua University. His research interests include sensor network, IoT, RFID, distributed systems, and cloud computing.

Feng Qian is an associate professor in the Ming Hsieh Department of Electrical and Computer Engineering, Viterbi School of Engineering at University of Southern California. His research interests cover mobile networking, AR/VR, wearable computing, real-world system measurements, and system security.

Zhao Zhang is a senior engineer in the AppInfra Research Center at ByteDance. He mainly focuses on automated and intelligent mobile app testing.

Ping Yang is the leader of the AppInfra Research Center at ByteDance. Her research interests are in software testing and AI for mobile infrastructure.

Tianyin Xu is an assistant professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His research focuses on techniques and tooling for the design and implementation of reliable and secure computer systems, especially those that operate at the cloud and datacenter scale.

REFERENCES

[1] B. Dolan-Gavitt, et al. 2011. Virtuoso: Narrowing the semantic gap in virtual machine introspection. *Proceedings of S&P.*
[2] L.K. Yan and H. Yin. 2012. DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic android malware analysis. *Proceedings of Security.*
[3] A. Machiry, R. Tahiliani and M. Naik. 2013. Dynodroid: An input generation system for Android apps. *Proceedings of FSE/ESEC.*
[4] S.R. Choudhary, A. Gorla and A. Orso. Automated test input generation for Android: Are we there yet? 2015. *Proceedings of ASE.*
[5] S. Rasthofer, et al. Making Malory behave maliciously: Targeted fuzzing of Android execution environments. 2017. *Proceedings of ICSE.*
[6] L. Fan, et al. Large-Scale analysis of framework-specific exceptions in Android apps. 2018. *Proceedings of ICSE.*
[7] H. Cai, et al. 2019. A large-scale study of application incompatibilities in Android. *Proceedings of ISSTA.*
[8] T. Gu, et al. 2019. Practical GUI testing of Android applications via model abstraction and refinement. *Proceedings of ICSE.*
[9] Z. Dong, et al. 2020. Time-travel testing of Android apps. *Proceedings of ICSE.*
[10] ByteDance. Douyin: Short Video Sharing Platform. https://www.douyin.com/
[11] Google. Google Android Emulator. https://developer.android.com/studio/run/emulator.
[12] D. Gao, et al. Trinity: High-performance mobile emulation through graphics projection. *Proceedings of OSDI.*
[13] I. Pustogarov, Q. Wu and D. Lie. Ex-vivo dynamic analysis framework for Android device drivers. *Proceedings of S&P.*
[14] Z. Lv, et al. Fastbot2: Reusable automated model-based GUI testing for Android enhanced by reinforcement learning. *Proceedings of ASE.*