Late Breaking Results: Towards Practical Record and Replay for Mobile Applications

Onur Sahin, Assel Aliyeva, Hariharan Mathavan, Ayse Coskun, Manuel Egele
ECE Department, Boston University, MA, USA
{sahin,aliyevaa,hrhrnm,acoskun,megele}@bu.edu

ABSTRACT

The ability to repeat the execution of a program is a fundamental requirement in evaluating computer systems and apps. Reproducing executions of mobile apps has proven difficult under real-life scenarios due to different sources of external inputs and interactive nature of the apps. We present a new practical record/replay framework for Android, RandR, which handles multiple sources of input and provides cross-device replay capabilities through a dynamic instrumentation approach. We demonstrate the feasibility of RandR by recording and replaying a set of real-world apps.

1 INTRODUCTION

Reproducing the execution of a program is a core requirement in many areas of computing such as workload characterization for computer architecture [15], system optimization for performance or energy [13] or software debugging and testing [6]. For instance, from the perspective of OS-level power/performance optimization, the same execution of a program is often repeated under different scheduling or power management policies to explore energy and performance tradeoffs. Meeting this core requirement, however, has proven challenging for mobile apps [10, 14], whose execution is heavily influenced by various non-deterministic factors such as user inputs, network or sensory input. Due to inability to easily reproduce real-life behavior of apps, prior studies analyze mobile workloads under limited usage scenarios (e.g., only app launch) [12] or rely on hand-crafted test scripts for specific devices and apps [11]. Systematic understanding of mobile workloads and comparison of experimental observations require the availability of tools to reproduce realistic executions in a cross-platform manner.

While various approaches exist for record and replay of mobile apps, several limitations impair their accuracy and practicality. First, most prior work [7–9] focus primarily on UI events and, thus, cannot handle execution variations that arise due to other inputs such as random numbers or, most dominantly, network events. Moreover, reliance on raw screen coordinates of UI interactions [8, 10, 14] restricts the replay capability to only one specific screen size. Second, prior approaches incur practical challenges as they require apps' source code [2, 4, 7] or perform intrusive modifications to the underlying OS or virtual machine [10]. Many real world Android apps are closed-source. Modifying the OS often requires access to proprietary code and is difficult to implement and maintain [14] due to the fragmented Android ecosystem with many software versions and custom devices [9]. In addition, due to manufacturer imposed restrictions against gaining root privileges, existing approaches have limited applicability to commercial off-the-shelf devices.

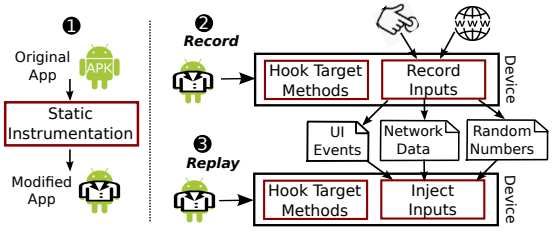

Figure 1: High-level view of RandR.

This paper presents RandR, which allows for timing-sensitive and cross-platform record and replay of multiple sources of inputs, while avoiding the restrictions that undermine practicality. Our primary insight is to record the inputs that drive the application execution by *runtime hooking* (as opposed to static OS or VM instrumentation) within an application's own virtual memory. RandR hooks into a set of target Java methods to capture both user inputs and the random numbers that cause variant application behavior if not kept deterministic across record and replay. In addition, RandR intercepts the network traffic by hooking into native C libraries. By capturing these multiple data and event streams and performing timing-sensitive and coordinate-independent replay of UI interactions, RandR successfully reproduces the behavior of popular off-the-shelf Android apps across different devices.

2 RANDR DESIGN

The goal of RandR (Figure 1) is to provide cross-device record and replay capabilities for real-world closed-source Android apps. RandR does not require root permissions or any modifications to OS/app source. Instead, RandR leverages Android application sandbox which ensures that each app runs in its own process space within its own instance of `Android Runtime`. By *dynamically* instrumenting Android Framework from within app's own virtual space, RandR is able to capture and reproduce the inputs to a set of target method calls (i.e., both `Java` and `C` APIs) in the framework.

App Instrumentation: RandR hooks into a set of target API methods to capture the inputs while recording and to inject the recorded inputs during replay. We use two open-source hooking frameworks, YAHFA[5] and AndHook[1] to hook into Java and native APIs respectively. RandR backs up the original target method and modifies the function pointers within the method instances to point to our hook functions. Our hook functions either record or modify method arguments and return values, and invoke the original target methods as necessary. To load our hooking library at runtime, RandR also statically re-writes the app bytecode.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3322476

App	Jaccard Similarity		Visual Similarity		App	Jaccard Similarity		Visual Similarity	
	RandR	Reran	RandR	Reran		RandR	Reran	RandR	Reran
Knights of Alentejo	100%	100%	Yes	Yes	Summation	100%	94%	Yes	No
Solar Compass	97%	97%	Yes	No	Accordion	100%	97%	Yes	Yes
Verbiste Android	97%	80%	Yes	Yes	Word Power	100%	98%	Yes	No
ToneDef	100%	100%	Yes	No	MedicLog	100%	41%	Yes	No
Mileage	100%	47%	Yes	No	Dicer	100%	94%	Yes	No

Table 1: Cross-device evaluation of RandR and Reran[8]

Widget-sensitive UI Replay: RandR's approach to UI replay is based on the observation that a user mostly interacts with an application via UI widgets, essential UI elements provided by the Android Framework (e.g. `Button`, `SearchBar`, `ImageView`, etc). Therefore, RandR targets widget-sensitive UI replay, and records the UI events (e.g., `MotionEvent`, `KeyEvent`) with respect to an app's UI widgets. To uniquely identify widgets presented in an app, RandR assigns widgets stable identifiers that are consistent across multiple application runs and devices. RandR derives these identifiers from multiple internal properties of a widget, including the widget's position in the UI layout, text and other internal fields. During the replay, RandR identifies the widgets on the screen, updates the recorded events according to the new location of a widget, and injects updated events into the app.

Network Replay: RandR records and replays the network I/O by hooking into the wrapper methods for various system calls (e.g., `socket`, `recv`, `write`) in Android's C library. While recording, RandR invokes the original library method and saves the arguments to a separate trace file for each socket descriptor. At the replay, once the `connect()` method is called on a socket, RandR identifies the target trace file to read based on the target IP address. By focusing on the system call interface, RandR is agnostic to different network protocol implementations in Java (e.g., OkHttp, FTPClient). In fact, RandR can even replay encrypted `HTTPS` traffic by intercepting and replaying the random numbers generated within crypto libraries during the `TLS` handshake.

Random Number Replay: Since random numbers can cause variant app behavior [10], RandR also handles this non-determinism by hooking into the pseudo-random number generator in Java APIs.

3 EVALUATION AND CASE STUDIES

We evaluate RandR's replay accuracy via two metrics: (1) Jaccard similarity between the set of executed methods of an app during record and replay; (2) user-visible state similarity as used in prior work [8–10]. We use a common `Java` code coverage tool (i.e., `EMMA` [3]) to obtain the set of executed methods. Since `EMMA` requires the app's source code for instrumentation, we crawl a random set of real-world Android apps from the F-droid dataset, and select 10 apps that `EMMA` and RandR can instrument without any errors.

We evaluate RandR's cross device replay capability on two Android SDK emulators (Nexus 5X and Pixel 2 XL) that have different screen sizes. Overall, when exercising applications during record, we achieve an average 49.45% method coverage. As shown in Table 1, RandR was able to successfully replay all apps across both devices, unlike the coordinates-based Reran tool.

To assess RandR's ability to record and replay closed-source apps, we pick the most popular apps with network dependent functionalities (i.e., using both `HTTP` and `HTTPS`) from 4 Play Store categories: OfficeSuite, Kakao Bus, Mirror Camera, Hot Pepper Gourmet. For these apps, we verified RandR's replay success by comparing the recorded and reproduced user-visible screen states.

(a) Recorded (b) RERAN (c) RandR

Figure 2: Comparison of Reran [8] to RandR for 2048 app

We show the importance of handling inputs other than UI (e.g., random numbers) with a specific study comparing RandR to Reran [8] for a session of the 2048 game (Figure 2). During record, the game reaches the "Game Over" state and renders a new text on the screen. Replay with Reran diverges to a different state due to randomized location of numbers, while RandR reproduces the same sequence of random numbers and reaches the correct final state.

Impact of Accurate Replay on Performance Measurements: RandR allows for real-life experimentations (e.g., power or performance studies) on mobile systems with off-the-shelf mobile apps and can substantially improve the quality of experimental measurements. Figure 3 illustrates the significance of replaying network traffic (e.g., using RandR), as opposed to UI-only replay [7–9, 14], for achieving consistent and meaningful performance measurements. We measured the latency distribution over 20 executions of `UC Browser` app from the Play Store while browsing a web page whose content changes over time (i.e., `thefakenewsgenerator.com`). Since RandR replays the same recorded content and is not effected by the network speed fluctuations or the web content changes on the server, RandR can significantly reduce measurement variations. Such reproducibility is key to correct analysis of performance and power bottlenecks in mobile systems research.

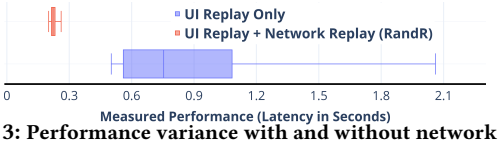

Figure 3: Performance variance with and without network replay

REFERENCES

[1] Andhook. https://github.com/asLody/AndHook.
[2] Android ui automator. https://developer.android.com/training/testing/ui-automator.
[3] Emma. http://emma.sourceforge.net.
[4] Espresso. https://developer.android.com/training/testing/espresso/.
[5] Yahfa. https://github.com/rk700/YAHFA.
[6] T. Azim and I. Neamtiu. Targeted and depth-first exploration for systematic testing of android apps. In *OOPSLA*, 2013.
[7] M. Fazzini, E. N. D. A. Freitas, S. R. Choudhary, and A. Orso. Barista: A technique for recording, encoding, and running platform independent android tests. In *ICST*, 2017.
[8] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein. Reran: Timing- and touch-sensitive record and replay for android. In *ICSE*, 2013.
[9] M. Halpern, Y. Zhu, R. Peri, and V. Janapa Reddi. Mosaic: cross-platform user-interaction record and replay for the fragmented android ecosystem. In *ISPASS*, 2015.
[10] Y. Hu, T. Azim, and I. Neamtiu. Versatile yet lightweight record-and-replay for android. In *OOPSLA*, 2015.
[11] X. Li, G. Chen, and W. Wen. Energy-efficient execution for repetitive app usages on big.little architectures. In *DAC*, 2017.
[12] D. Pandiyan, S. Lee, and C. Wu. Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite - mobilebench. In *IISWC*, 2013.
[13] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. Integrated cpu-gpu power management for 3d mobile games. In *DAC*, 2014.
[14] Z. Qin, Y. Tang, E. Novak, and Q. Li. Mobiplay: A remote execution based record-and-replay tool for mobile applications. In *ICSE*, 2016.
[15] D. Sunwoo, W. Wang, M. Ghosh, C. Sudanthi, G. Blake, C. D. Emmons, and N. C. Paver. A structured approach to the simulation, analysis and characterization of smartphone applications. In *IISWC*, 2013.