Poster: Efficient and Deterministic Replay for Web-enabled Android Apps

Fangge Yan
Zhengwei Qi
thunder_s@sjtu.edu.cn
qizhwei@sjtu.edu.cn
Shanghai Jiao Tong University

Mingyuan Xia
Xue Liu
mingyuan.xia@mail.mcgill.ca
xueliu@cs.mcgill.ca
McGill University

ABSTRACT

Mobile apps are adopting web techniques for improved development agility. In this paper, we propose TimelyRep to help mobile developers debug and test their web-enabled Android apps. TimelyRep provides efficient deterministic record-and-replay as a software library, running on unmodified Android. Also, as touchscreen becomes the major interaction method for mobile devices, web-enabled apps can receive many events in short periods. TimelyRep embodies a mechanism to control replay delays and achieve smooth replay. TimelyRep also supports cross-device replay where the event trace captured on one device can be replayed on another. We evaluate TimelyRep with real-world web applications. The results show that TimelyRep is useful for reproducing program bugs and has higher timing precision than previous tools.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**;

KEYWORDS

Android, deterministic replay, mobile web app, software testing

1 INTRODUCTION

Mobile developers use web techniques to deploy new contents and bug fixes instantaneously. Web replay tools can reproduce same execution and are used to test and debug web apps [1, 3, 4]. Such tools capture events that can affect web program state at the recording stage and inject same synthetic events back at the replaying stage. An effective replay tool must ensure deterministism (the same program states between the recording and replaying stage) and mitigate replay delays (synthetic events issued with similar timing as the original events). A timing-precise replay can help developers find timing-related vulnerabilities [2] and tune performance.

To ensure deterministism, previous approaches require an external proxy server [3] or modification to the browser [1, 4], which are not always available for the mobile usage scenarios. Also, due to the widespread use of touchscreen, the intervals between two consecutive recorded events for the mobile scenarios are dramatically shorter than desktop cases. Consequently, a replay tool often needs to emit a large number of synthetic events in a short period and thus replay delays become a crucial problem.

In this paper, our contribution is three-fold:

- We develop a mechanism to control replay delays in the JavaScript space. This mechanism can function for both the mobile web embeddings and traditional web browsers.
- We develop a mechanism to deliver HTTP responses deterministically. This mechanism functions purely within the mobile application itself.
- We propose TimelyRep, a web replay tool that supports deterministic and efficient web replay for mobile apps. TimelyRep embodies the above two mechanisms we developed to achieve deterministic replay and high replay timing precision. TimelyRep also supports cross-device replay.

2 TIMELYREP'S DESIGN

2.1 Overview

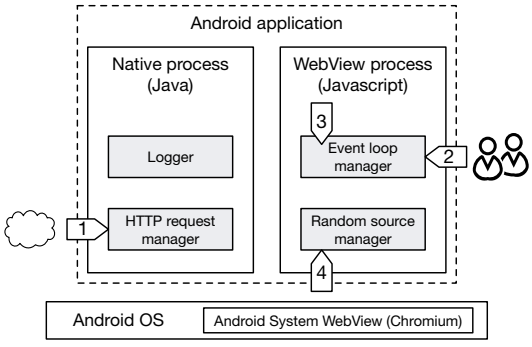


Figure 1: TimelyRep's architecture

TimelyRep records various non-deterministic events in web-enabled apps and replays them back with low delay for single device and across devices. TimelyRep is provided as a software library and integrated with a web-enabled Android application by mobile developers. TimelyRep runs in both the native process (Java) and the WebView process (JavaScript) of an app. Figure 1 shows the architecture of TimelyRep and its four components in the two processes. All components work in the application space and require no extra Android permission or modification to the underlying Android components.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194994

Four non-deterministic sources are managed. HTTP requests (source 1), including the response data and the network latency, will pass through the HTTP request manager in the native process. These requests would be recorded at recording stage and replayed back exactly at replaying stage. User inputted DOM events (source 2) are user interactions, which will be intercepted and logged by the event loop manager. Callback functions (source 3) are functions scheduled by Web APIs (e.g. `setTimeout()`) from the application's JavaScript code. Such callbacks are also managed by the event loop manager. Finally, random sources, such as `Random()` and `Date()`, are intercepted by the random source manager to ensure that the application receives the same values during replaying. The two managers in the WebView process provide deterministic replay services for a single page frame. The logger in the native process merges events from multiple page frames and creates a unified event trace for whole-application replay.

2.2 Replay Timing Control Mechanism

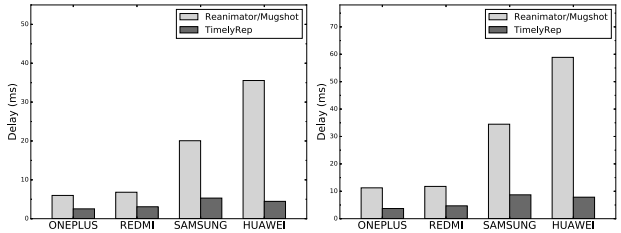
During the recording stage, TimelyRep logs a sequence of events in chronological order ($T_0 \cdots T_N$). During the replaying stage, TimelyRep schedules these events according to the log trace. Every time a scheduled event is fired, TimelyRep decodes the event content and performs the replay action. Suppose the replay starts at t_0, the n-th event should happen at $t_0 + (T_n - T_0)$. When an event (e.g., the n-th event) is replayed at t_n, TimelyRep calculates the relative delay $t_n - (t_0 + (T_n - T_0))$ to denote the replay *delay*. A zero delay denotes perfect timing and indicates the event is fired as schedule. However, a scheduled event may be postponed by other tasks, causing a positive delay. For example, `setTimeout` is usually used to schedule a callback function after a certain time period. And it could be postponed when random pauses (e.g., DOM rendering pause, Garbage Collection pause) happen.

In order to reduce the replay delay, TimelyRep gives a scheduled event (e.g., the n-th event) priority according to its *schedule_interval*, i.e. $t_0 + (T_n - T_0) - now$. The event with shorter schedule interval is given higher priority, making it cannot be postponed. When an event is already late ($schedule\_interval < 0$), TimelyRep executes the event directly in current call stack. When an event has larger interval, TimelyRep chooses the normal `setTimeout`. When an event has small interval, TimelyRep uses `postMessage` as a higher priority way. However, `postMessage` usually performs as an immediately scheduling. Considering that `setTimeout` cannot create a zero-delay execution, i.e., `setTimeout(f, 0)` would execute after an inevitable delay (bounded in 4ms but varying across devices), TimelyRep selects 2ms (half of the `setTimeout` delay bound) as a threshold, which results in a replay delay range of $-2ms$ to $+2ms$. Specifically:

$$\begin{cases} direct, & schedule\_interval < 0 \\ postMessage, & 0 \le schedule\_interval < 2 \\ setTimeout, & schedule\_interval \ge 2 \end{cases}$$

3 EVALUATION

We evaluated TimelyRep with 10 real-world web apps on 6 different Android devices (from Android 4.4.4 to 7.1.1) to understand replay determinism, timing precision and cross-device replay.



(a) Average delay **(b) 80th percentile delay**

Figure 2: Timing precision comparing with Reanimator/-Mugshot.

Replay Determinism. We use 8 web applications with representative race conditions to evaluate replay determinism. These applications have different program states across different runs because of various non-deterministic sources and their occurring order. TimelyRep reproduces these bugs and program states.

Timing Precision. Games involve intensive user interactions and thus stress both the correctness and the timing precision. We select 2 game apps (MineSweeper and Tetris) to evaluate replay timing precision. The delay values are limited in 13ms for most events when we replay a high-event-rate log in a median class device, Xiaomi Redmi Note 2. We also compare TimelyRep with Mugshot/Reanimator[1] by providing the same recorded traces to them to evaluate their timing precision. Figure 2 shows that with the delay control mechanism and properly choosing different event scheduling methods, TimelyRep's replay delay is 2 to 6 times shorter than Reanimator.

Cross-Device Replay. We conduct cross-device replay by replaying a trace from a fast device to a slow device, confirming that TimelyRep still has high timing precision in cross-device replay.

ACKNOWLEDGMENTS

This work was supported in part by National Infrastructure Development Program (No. 2013FY111900), National NSF of China (No. 61672344), and Shanghai Key Laboratory of Scalable Computing and Systems.

REFERENCES

[1] Brian Burg, Richard Bailey, Andrew J. Ko, and Michael D. Ernst. 2013. Interactive Record/Replay for Web Application Debugging. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 473–484. https://doi.org/10.1145/2501988.2502050

[2] Ang Chen, W. Brad Moore, Hanjun Xiao, Andreas Haeberlen, Linh Thi Xuan Phan, Micah Sherr, and Wenchao Zhou. 2014. Detecting Covert Timing Channels with Time-deterministic Replay. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Berkeley, CA, USA, 541–554. http://dl.acm.org/citation.cfm?id=2685048.2685091

[3] James Mickens, Jeremy Elson, and Jon Howell. 2010. Mugshot: Deterministic Capture and Replay for Javascript Applications. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI'10)*. USENIX Association, Berkeley, CA, USA, 11–11.

[4] Christopher Neasbitt, Bo Li, Roberto Perdisci, Long Lu, Kapil Singh, and Kang Li. 2015. WebCapsule: Towards a Lightweight Forensic Engine for Web Browsers. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 133–145. https://doi.org/10.1145/2810103.2813656

[1] Mugshot is not publicly available while Reanimator is an open-source implementation of Mugshot.