软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.005799]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种基于录制/重放的 Android 应用众包测试方法*

曹羽中[1,2], 吴国全[1,3,4], 陈 伟[1,3,4], 魏 峻[1,3,4], 黄 涛[1,3,4], 王 溯[2]

[1](中国科学院 软件研究所 软件工程技术中心,北京 100190)

[2](北京城市学院 信息学部,北京 100083)

[3](计算机科学技术国家重点实验室(中国科学院 软件研究所) ,北京 100190)

[4](中国科学院大学,北京 100049)

摘 要: 随着 Android 设备的流行和普及,Android 生态系统的碎片化问题越发严重.为确保应用质量,Android 应用需要在多种设备上进行测试.为了应对大量重复机械的测试工作,学术界和工业界提出了众多跨设备的测试方法,但目前的方法还有较多的局限性：1)手工编写设备无关的测试脚本耗时且容易出错;2)现有录制/重放方法生成的测试脚本在跨设备重放时会出现各种问题,导致重放失败;3)由于缺少足够的 Android 设备,应用难以在大量不同类型的设备上进行测试;4)现有的测试方法由于缺少应用特定的领域知识.无法生成有效的用户输入,导致测试覆盖率不高.基于以上原因,大量的应用在没有经过充分测试后发布,兼容性问题频发.针对以上问题, 本文提出了一种基于录制/重放的 Android 应用众包测试方法,并实现了原型工具 AppCheck. AppCheck 收集众包用户和设备交互时所产生的事件序列后,将其转换为平台无关的测试脚本,可直接在众包用户的设备上进行重放.在重放期间, AppCheck 收集各种测试相关数据(例如,截图和布局信息)以检测兼容性问题.实验结果表明,AppCheck 能够有效的完成跨设备录制/重放以及兼容性问题的检测,改进了当前方法的不足.

关键词: 安卓;众包测试;碎片化;自动化测试;录制;重放

中图法分类号: TP311

中文引用格式: 曹羽中,吴国全,陈伟,魏峻,黄涛,王溯. 一种基于录制/重放的 Android 应用众包测试方法.软件学报. http://www.jos.org.cn/ 1000-9825/5799.htm

英文引用格式: Cao YZ, Wu GQ, Chen W, Wei J, Huang T,Wang S. Crowdsourcing test method for Android applications based on recording/replay. Ruan Jian Xue Bao/Journal of Software, (in Chinese). http://www.jos.org.cn/1000-9825/5799.htm

Crowdsourcing test method for Android applications based on recording/replay

CAO Yu-Zhong[1], WU Guo-Quan[123], CHEN Wei[123], WEI Jun[123], HUANG Tao[123], Wang Su[2]

[1](Technology Center of Software Engineering, Institute of Software, CAS, Beijing 100190, China)

[2](Department of Information, Beijing City University, Beijing 100083, China)

[3](State Key Laboratory of Computer Science, Institute of Software, CAS, Beijing 100190, China)

[3](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: It is well known that the fragmentation of Android ecosystem has caused severe compatibility issues. Therefore, for Android apps, cross-platform testing (the apps must be tested on a multitude of devices and operating system versions) is particularly important to assure their quality. Although lots of cross-platform testing techniques have been proposed, there are still some limitations: 1) it is time-consuming and error-prone to encode platform-agnostic tests manually; 2) test scripts generated by existing record/replay techniques are brittle and will crash when replayed on different platforms; 3) Developers, and even test vendors have not equipped with some special Android devices. 4) Due to the lack of specific domain knowledge, the existing test methods cannot generate effective user inputs, resulting in low testing coverage. As a result, apps that have not been fully tested, will lead to many compatibility issues after releasing. To address these limitations, this paper proposes AppCheck, a crowdsourced testing service for Android apps. To generate tests that will explore different behavior of the app automatically, AppCheck crowdsources event trace collection over the Internet, and various touch events will be captured when real users interact with the app. The collected event traces are then transformed into device-independent test scripts, and directly replayed on the devices of real users. During the replay, various data (e.g., screenshots and layout information) will be extracted to identify compatibility issues. Our empirical evaluation shows that the proposed AppCheck is effective and improves limitations of the state of the art.

Key words: Android; Crowdsourced testing; Fragmentation; Automated Test; Recording and Replay

* 基金项目: 重点研发计划(2017YFA0700603);国家自然科学基金面上项目(61472407);北京城市学院科研种子基金(KYZZ201801);北京城市学院 2018 年度实培计划项目

Foundation item: National Key Research and Development Plan (2017YFA0700603);National Natural Science Foundation of China(61472407);Scientific Research Projects of Beijing City University(KYZZ201801);The 2018 annual Graduation Practice Training Program of Beijing City University

收稿时间: 2018-05-07; 修改时间: 2018-06-21, 2018-09-27; 采用时间: 2018-12-06; jos 在线出版时间: 2019-04-10

CNKI 网络优先出版: 2019-04-11 09:52:47, http://kns.cnki.net/kcms/detail/11.2560.TP.20190411.0952.002.html

随着移动互联网技术的发展,未来互联网重心从传统的个人电脑转移到了新一代移动设备上,智能手机开始被广泛使用.我们的日常生活,如购物,银行和旅行等活动都离不开手机应用程序.特别是 Google 公司推出移动智能设备操作系统 Android 之后,各大主流手机制造商大规模地推出 Android 智能手机,Android 设备种类从 2014 年的 18796[1]增长到 2015 年的 24093[2].在 2017 年 Google I/O 大会上,谷歌宣布每月活跃安卓设备数量超过 20 亿[3].由于 Android 系统版本更迭频繁,同时为了吸引用户并在竞争中保持优势,不同的设备厂商会对 Android 系统进行定制,从而导致 Android 生态系统的碎片化.碎片化现象给 Android 应用的测试带来了巨大挑战,应用开发者需要在多种不同类型、不同版本设备上进行大量重复机械的测试工作,Android 应用的自动化测试技术因此成为研究热点.

在学术界,虽然目前已经有很多针对 Android 应用的自动化测试技术和工具[4-6],然而,根据最近的实证研究[7],这些技术的测试覆盖率仍然很低,这是因为大量的应用程序是为具备特定领域知识用户开发的,如果不具备这些领域知识,将无法触发某些特定的交互场景.在工业界,已有多种 Android 应用自动化测试框架被广泛使用,如 Android Espresso [8], Robotium [9],UIAutomator [10],Appium [11]等.这些框架可以跨设备自动执行测试用例(Appium [11]甚至支持测试 Hybrid App).但是不足在于需要测试人员掌握特定的编程语言,并且手工编码会消耗测试人员大量的精力.

用户交互行为录制/重放是一种常见的 Android 应用测试方法[12] [13].测试人员通过记录收集用户与被测应用的交互事件序列,并在不同设备进行重放[47],这种方式可以用于 Android 应用的兼容性测试,实现测试成本的降低和测试效率的提升[14][15].现有的录制/重放技术[12]利用 Android 底层接口获取系统日志,进而形成测试脚本,具有不侵入应用内部、可以描述多应用交互等优点,但得到的原生事件流脚本接近于机器码,不便于测试人员进行编辑修改;并且生成的测试脚本难以实现跨设备重放,缺乏重用性.尽管相关工作尝试采用成线性比例缩放机制实现跨设备屏幕尺寸和分辨率的适配[13],但方法适用性非常有限,当应用的界面布局随着屏幕旋转或屏幕尺寸、分辨率发生变化后,该方法会失效.这导致生成的测试脚本却不够健壮,在其他测试设备平台上运行时会崩溃.

区别于传统的 GUI 程序测试,Android 生态系统的碎片化要求应用需要在不同类型、不同版本的 Android 设备上进行测试,以保障应用的健壮性.但是在当前 Android 设备快速更新升级的情况下,应用开发者甚至专业的测试厂商[31]都不可能购买所有需要测试的 Android 设备.为了解决这一问题,众多测试厂商(如 uTest [16]和 Testin[17])使用了一个新兴的技术——众包测试[18]来测试移动应用.在众包测试中,拥有不同移动设备和使用环境的真实用户测试同一应用软件.如发现 Bug,用户将根据 Bug 的严重程度得到奖励.然而,这样的过程仍然需要大量繁琐的手工工作,同时用户报告的 Bug 质量得不到保证[19].通常情况下,Bug 在开发人员和用户之间多次沟通后才能得到确认.

针对以上不足,本文提出了一种基于录制/重放的 Android 应用用户交互行为跨设备众包测试方法,并基于该方法实现了录制重放工具 AppCheck.AppCheck 具备以下三个特点:

(1) 基于测试框架 STF[20],AppCheck 允许测试人员通过 PC 或智能手机浏览器和真实用户的设备进行交互,并在交互过程中捕获用户的各种 Touch 事件;
(2) 将设备相关的 Touch 事件转换为设备无关的抽象用户动作,并基于 Android 辅助功能[21]支持直接在参与众包测试的用户真机上进行重放.
(3) 通过收集在不同设备上重放产生的数据,设计兼容性问题检测算法,除了识别常见的兼容性问题之外还能识别不同设备重放时产生的性能问题(如按钮长时间不响应点击事件).

本文提供的方法还有以下优势:支持一次录制,处处运行.测试人员可以在某一特定设备捕获用户的操作序列后,将操作序列转换为设备无关的测试脚本后在其他设备进行重放;此外,本文介绍的方法不需要侵入应用进行插桩或对应用进行重签名.在录制阶段,AppCheck 仅需要用户通过浏览器操作应用便可以实现操作序列的收集工作.在重放阶段,AppCheck 会负责将转换后的设备无关操作序列分发到需要测试的用户设备上并完成重放.

为了实现以上目标,AppCheck 在实现上需要解决如下问题.首先,当用户与 App 交互时,产生的 Android 系统底层操作日志流依赖于特定平台,不能针对具有不同屏幕尺寸和分辨率的设备进行跨设备重放.为了解决这一问题,AppCheck 设计转换算法将捕获的低级事件抽象转化成平台无关的用户动作(如单击一个特定的资源 ID 按钮).同时,为了定位相应的 UI 组件,设计了三种不同的 UI 组件定位方式:资源 ID、属性和基于 XPath 的定位器,这些方式独立于设备的屏幕大小和分辨率,可以跨设备对 UI 组件进行定位.

为了检测应用在具有不同屏幕尺寸和分辨率设备上的兼容性问题,AppCheck 首先比较不同设备是否具有相同的 UI 组件布局.当组件布局相同时,AppCheck 通过对比不同设备的截图来进一步检测是否存在兼容性问题.除此之外,AppCheck 通过比较不同设备上用户操作的执行时间来识别性能相关的兼容性问题.本文的主要创新与贡献如下:

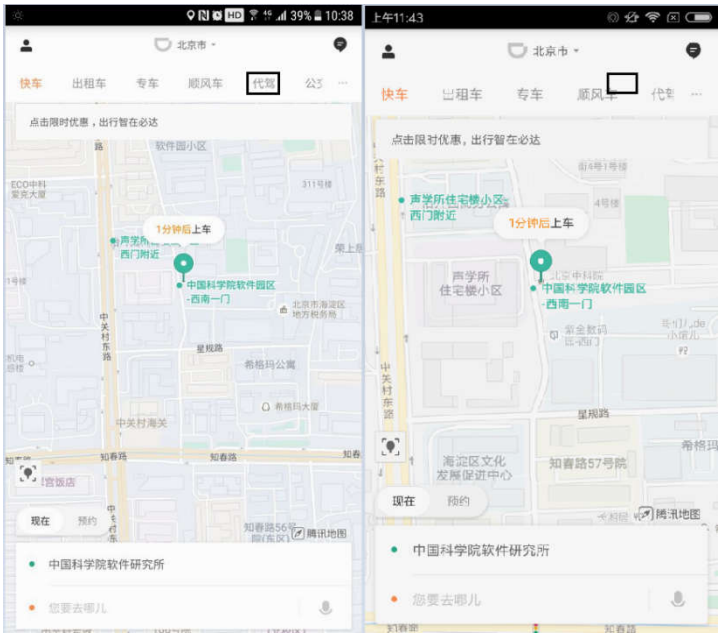
(1) 设计并实现了一种基于录制/重放的 Android 应用众包测试方法,并实现了工具原型 AppCheck.该工具通过众包测试的方式收集用户操作序列,并可直接在用户设备上重放所收集的操作序列.
(2) 设计了一种转换算法,支持将收集到的设备相关 Touch 事件转换为设备无关的抽象用户动作, 并基于辅助功能服务完成跨设备重放.

(3) 设计了一种 Android 应用兼容性问题的检测机制,以识别 Android 应用在不同设备重放时所发生的兼容性问题.

本文接下来的组织结构如下：第一部分给出了本文工作的研究动机及示例.第二部分给出了本文工作整体方法的详细说明.第三部分介绍了本文提出方法的具体实现细节.第四部分通过对各大应用商店挑选出的 100 个应用进行评估,以验证本文提出方法的有效性.第五部分介绍了本文方法的局限性.第六部分和第七部分是相关工作以及总结.

1 研究动机

我们以两个常用的移动应用为例子,说明现有录制/重放方法在跨设备重放时存在的不足.滴滴出行[22]是目前非常流行的移动应用,为用户提供了一个覆盖出租车、专车、快车、顺风车、代驾及大巴等多项业务在内的一站式出行平台.图 1 分别是滴滴出行((v5.1.20_284)在三星 Note5(Android 6.0)和红米 4X(Android 6.0)上运行的截图.可以发现代驾菜单位置在两个设备上有很大的差别.三星 Note5 中代驾菜单可以正常显示(如图 1(a)所示),而代驾菜单在红米 4X 中不能完全显示(如图 1(b)所示),甚至与三星 Note5 中代驾菜单的位置不对应.现有的记录/重放技术[12] [13] 无法处理这一场景,因为这些技术基于界面控件元素在跨设备时成线性比例缩放的假设.然而,这样的假设在该应用中并不成立,如下图 1,捕获的 Touch 事件(如点击代驾菜单)从三星 Note5 转换到红米 4X 上会失败,经过我们的实验,发现存在大量的移动应用界面控件元素在跨设备平台时并不成线性比例缩放.



(a) (b)

Fig.1 Didi Chuxing App

图 1 滴滴出行(代驾菜单等比例转换结果)

京东 Android 客户端[23]也是目前流行的一个移动应用程序,它是一款专为 Android 设计的手机购物软件,拥有商品购买、查询订单、订单跟踪、商品晒单等一站式购物平台.能够在多种 Android 设备(如华为、三星等)运行流畅.然而对于某些特定设备,例如在安装了小米 MIUI8 的设备上运行京东 Android 客户端(v7.0.6)时,却有明显的卡顿现象[24],为了发现类似的兼容性性能问题,购买所有需要测试的设备是不可实现的,一个可行的办法是使用众包测试方法,邀请拥有各种不同 Android 设备的用户,直接在他们的设备上利用进行测试.

综上,现有的 Android 录制/重放技术[12] [13] 假定 UI 控件在不同设备上成线性比例缩放,难以适用于跨设备重放,且依赖于 Android SDK 工具包提供的 adb 工具,需要设备在测试前先和服务器端连接,无法支持众包测试.同时,现有的兼容性问题检测技术[25] [26] [27]只能识别一些布局以及用户操作行为相关的问题,没有提供与应用运行性能相关的兼容性问题检测机制.

2 整体方法

为了解决现有技术框架的不足,我们提出了一种基于录制/重放的 Android 应用众包测试方法,并开发了相应的工具原型 AppCheck.AppCheck 借助 STF 框架[20]捕获真实用户和 App 交互时所产生的事件序列,并将这些事件序列转换成设备无关的中间语言测试脚本.该脚本可在参与众包测试的真实用户设备上直接进行重放,在重放过程中 AppCheck 收集截图和页面布局等信息,并使用检测算法进行兼容性问题的检测,最终向开发者提供可视化检测报告.图 2 展示了 AppCheck 的总体结构,主要包括四个步骤：事件序列收集、用户行为抽象,跨设备重放和兼容性问题检测.接下来,将介绍每一个步骤的具体实现细节.

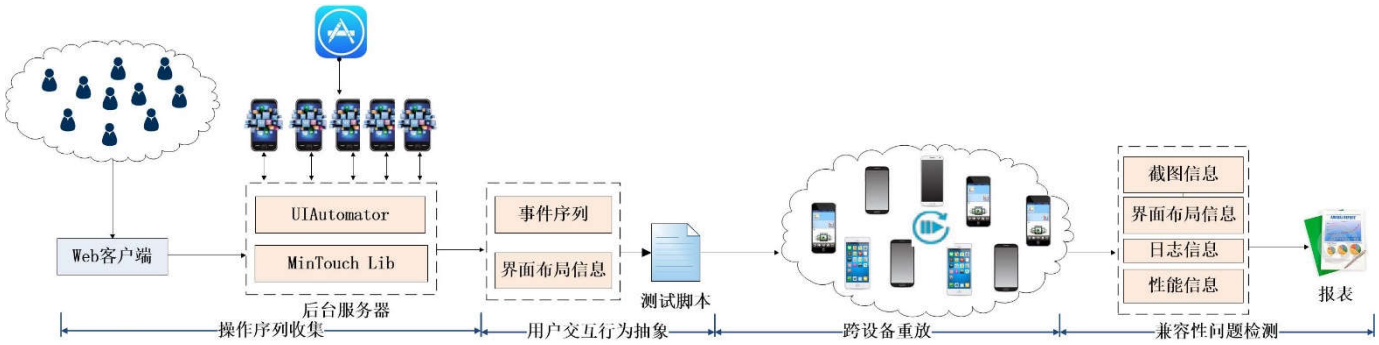


Fig.2 AppCheck Overview

图 2 AppCheck 总体结构

2.1 Android操作序列的收集

为了便于收集众包测试用户与 App 的交互事件序列,AppCheck 借助 STF 框架为用户提供了 Web 接口,通过该接口用户可与被测试应用进行交互.在交互过程中,各种用户操作将被捕获并记录.在该阶段,AppCheck 还负责收集每一个操作后应用的界面布局和 UI 组件信息.在用户交互行为抽象阶段,通过分析收集到的界面布局和 UI 组件信息,AppCheck 将捕获到与设备相关的日志流信息转换为与设备无关的用户操作序列.

为了支持事件序列的收集,AppCheck 主要由三部分组成:被测 Android 设备平台,后端服务器,以及 Web 客户端.被测 Android 设备平台可以是真实的 Android 设备,也可以是 Android 模拟器.如果是真实的 Android 设备,后端服务器将通过互联网远程连接待测试设备.如果是模拟器,后端服务器将自动化配置模拟器参数并在后端服务器上运行模拟器.

后端服务器负责管理所有可连接的设备和模拟器.通过 Socket 连接,后端服务器持续不断向 Web 客户端发送被测试应用的运行状态截图和 UI 信息.同时后端服务器还负责将用户的操作序列从 Web 客户端传递到待测试设备.

Web 客户端可实时在浏览器显示被测试应用在测试设备上运行的 JPEG 格式截图,同时将用户在浏览器上的操作序列发送到测试设备上执行.Web 客户端可以运行在笔记本电脑、台式机、甚至移动智能设备上.后端服务器和 Web 客户端的实现都基于开源的 Open STF[20]框架,该框架整合了 minitouch 库[28],minitouch 提供了一个 Socket 接口用来收集 Android 设备上的多点触控事件.STF 框架支持 Android API Level 10 以上的设备且不需要 Root 权限.通过整合该框架 AppCheck 能够支持触屏手机常规的手势操作,如 MultiTouch 和 Swipe 等手势操作.

当用户通过 Web 客户端操作被测应用的界面时, Touch 事件和坐标信息会被 minitouch 捕获并通过网络传输到 Android 设备上.在 minitouch 中,一个 Touch 事件被定义为:

type <contact> <x> <y> <pressure>

其中 type 描述了 Touch 事件的类型,它的取值可以是 d(表示 touch down 事件), m(表示 move on 事件)以及 u(表示 touch up 事件) ,<contact>表示多点触控手势操作中手指的标示值,< x > < y >存储被触碰的坐标.<pressure>表示该触控事件的力度,其中< x > < y >值和<pressure>值均可为 Null.

除了 Touch 事件,minitouch 还有一些其他类型的事件.例如,c 是一个提交事件,它表示提交当前操作序列的一次提交.在同一个提交中,同一根手指不能有一个以上的 d、m 或 u 事件.当 c 事件被提交到测试设备后,minitouch 将执行在屏幕上触发当前的操作集合.例如,当测试人员出在 Web 客户端触发一个 touch down 事件后,生成操作序列<d 0 10 10 50>,<c>,表示一根手指以 50 的压力在屏幕坐标(10,10)处按下. 该操作序列将在提交事件 c 到达后被执行.

然而,捕获的 Touch 事件序列包含设备相关的坐标信息,无法直接在不同分辨率和不同屏幕尺寸的设备上重放.为了解决这一问题,在此阶段,AppCheck 还保存 Touch 事件发生后被测应用的 UI 结构信息. 借助于收集的 UI 结构信息,设备平台相关的 Touch 事件序列将被转换成为平台无关的用户操作序列.

在 Android 平台上,应用的 UI 组件结构信息可以通过 Hierarchy Viewer[29]以及 UIAutomator[30]等工具获取.因为 Hierarchy Viewer 需要在 Root 模式下工作,我们选择 UIAutomator 作为 UI 组件结构信息的获取工具.然而,UIAutomator 服务默认将 UI 组件信息储存在外部存储设备上,同时生成记录 UI 组件信息的文件也需要数秒,无法达到实时收集 UI 组件的目标.为了解决以上问题,我们修改了 UIAutomator,精简了 UI 组件属性的数量,修改后的 UIAutomator 只记录 UI 组件的资源 ID 和 index、文本内容、坐标范围等信息.在修改后,UI 组件信息文件的生成仅仅需要数百毫秒.

2.2 用户行为抽象

为了支持跨设备重放,AppCheck 将用户 Touch 事件序列转化为可在各种类型 Android 设备上执行的平台无关中间脚本语言,其文法定义如下:

Table 1 Syntax of device-agnostic representation

表 1 中间语言语法

中间语言语法

pgrm ::= *(stmt)**
stmt ::= timestamp *action selector*
action ::= **Click** | **LongClick** | *InputValue* | *SwipeType* | *ScrollType* | *MutiTouchType* | *EntityKey*
InputValue ::= **Input** *value*
value ::= string
SwipeType ::= **Swipe-up** | **Swipe-down** | **Swipe-right** | **Swipe-left**
ScrollType ::= **Scroll-forward** | **Scroll-backward**
MutiTouchType ::= **zoom-in** | **zoom-out**
EntityKey ::= **home** | **menu** | **back**
selector ::= **resources-id** | *resources-id-property* | *XPath*
resources-id-property ::= **resources-id** *property*
property ::= *index* | *className*
index ::= number
className ::= string
XPath ::= string

中间脚本语言包含用户操作以及所操作的 UI 组件(通过 selector 定位).目前,AppCheck 定义了七种用户的操作：Click、LongClick、Input、Swipe、Scroll、MultiTouch 以及 EntityKey(实体键).为了精确识别平台无关的操作,AppCheck 定义用户的每个操作以 Press 事件开始,中间包括零个或多个 Move 事件,最终以 Release 事件结束(对于 MultiTouch 这样的手势操作,会有一个 Press 事件,一个以上的 Move 事件和一个 Release 事件组成).同时对每个操作定义了于之相对应的时间自动机.下面给出 AppCheck 所使用的时间自动机定义.

定义 1 AppCheck 中使用的时间自动机是一个六元组$\mathbf{M}=(\mathbf{S},\mathbf{\Sigma},\boldsymbol{S_0},\mathbf{C},\mathbf{E},\mathbf{F})$,其中$\mathbf{S}$为一个有限状态集合,$\mathbf{\Sigma}$为集合$\{\mathbf{c},\mathbf{d},\mathbf{m},\mathbf{u}\}$,其中 c, d, m, u 是 minitouch 中所定义的四种事件的类型.$\boldsymbol{S_0}\in\mathbf{S}$为初始状态,$\mathbf{C}$是表示时钟集合,该集合的元素个数是有限的,$\mathbf{E}\subseteq\mathbf{S}\times\mathbf{\Sigma}\times\mathbf{\Phi(C)}\times\mathbf{2^C}\times\mathbf{S}$是边的集合.其中$\mathbf{\Phi(C)}$表示时钟约束的集合,边$<\boldsymbol{S_0},\mathbf{d},\boldsymbol{\delta},\boldsymbol{S_1}>$表示当输入字符串为$\mathbf{d}$时从状态$\boldsymbol{S_0}$到状态$\boldsymbol{S_1}$的转换.$\boldsymbol{\delta}$是时钟集合上的一个时钟约束,即$\boldsymbol{\delta}\boldsymbol{\epsilon}\mathbf{\Phi(C)}$,当$\boldsymbol{\delta}$满足时,转换才能发生,其中$\boldsymbol{\delta}$可以为空.$\mathbf{F}\subseteq\mathbf{S}$为$\mathbf{S}$的终止状态.

根据定义 1,Click 与 LongClick 操作的时间自动机如下图 3 所示,其中eLapsed 是时钟集合中的一个元素,eLapsed<500ms 和 eLapsed>500ms 是时钟集合上的两个时钟约束, eLapsed用于记录从S_0开始到达S_2状态所经历的时间.根据 2.1 中 minitouch 对 Click 事件的定义,S_0和S_2分别对应 Press 事件和 Release 事件的开始.根据 Android 系统默认的设定,当 Press 事件和 Release 事件操作码的时间间隔大于 500 毫秒[10]时,AppCheck 会认定当前操作序列是一个 LongClick 操作.例如下图 3 中 S_5为 Click 事件序列终点, S_6为 LongClick 事件序列终点：

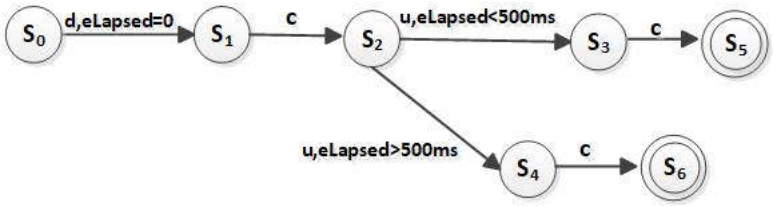


Fig.3 Click and LongClick Event timed automata
图 3 Click 和 LongClick 时间自动机

相对于识别 Click 操作,Input 事件的识别更加复杂,这是因为对于触屏智能手机,用户的输入都是依靠软键盘,这导致 Input 事件本身也是一个点击事件,但不同在于点击的目标是软键盘.为了识别 Input 事件,当用户每次点击可输入的 UI 组件(如 EditText)后, AppCheck 会检验可输入的 UI 组件是否成为屏幕焦点,如果用户点击可输入的 UI 组件且其 text 属性发生变化,AppCheck 有限自动机将判定当前为一个 Input 事件,通过查找 UI 结构树(Android 的 UI 组件结构信息是以树形结构组织)获取组件的 Text 属性,当组件失去焦点后,AppCheck 会判定输入完成,读取所操作组件的 Text 属性值作为最终输入.Input 事件的识别算法如下：

算法 1: Input 事件的识别算法 IEI(Input event identify)

输入：ES_i：操作事件　control：UI 组件
输出：InputStmt：平台无关的字符串输入语句

```
01：If typeof (ES_i) is Click event then
02：    If typeof(control) is android.widget.EditText then
03:       index ← control .index
04:       resourceid ← control .ResourceId
05：       While control is forced do
06:            bufString ← control.text
07：       End While
08:     End If
09:   End If
10:   InputStmt ← Input resourceid index bufString
11: return InputStmt
```

IEI 算法的主要输入有:操作事件 ES_i,UI 组件 control.算法输出是平台无关的字符串输入语句 InputStmt.在 IEI 算法的第 1-2 行中,首先判断当前操作事件 ES_i 是否是 Click 事件且 UI 组件类型为 android.widget.EditText 控件,如果满足这两个条件, AppCheck 将会认定当前操作为输入操作. IEI 算法的 3-4 行负责记录输入内容以及控件 ResourceID、index 等信息.在算法的 5-11 行中,AppCheck 根据当前控件是否获得焦点来判断输入字符串是否输入完毕.当前控件失去焦点后,算法第 10-11 行会判定输入结束并返回平台无关的输入语句 InputStmt 以供重放使用.

Scroll 事件主要应用于控件 android.widget.ScrollView,其自动机以(d c)开始,而后伴随着多个(m c),最后以(u c)结束. 根据定义 1,Scroll 事件序列的时间自动机如下图 4 所示(其中S_6代表 Scroll 或 Swipe 事件识别成功).识别 Scroll event 后,AppCheck 将根据起始点与终点位置关系来判断用户手指的移动方向,进一步将 Scroll 划分为两类：Scroll-forward 以及 Scroll-backward.

Swipe 事件的自动机与 Scroll 事件相同,区别是当前事件序列是否应用于 android.widget.ScrollView 控件.如果是,AppCheck 会判定当前事件为 Scroll 事件,否则将判定当前事件序列为 Swipe 事件.同样的,根据用户手指的移动方向 AppCheck 进一步将 Swipe 划分为四类：Swipe-up、Swipe-down、Swipe-right 以及 Swipe-left.

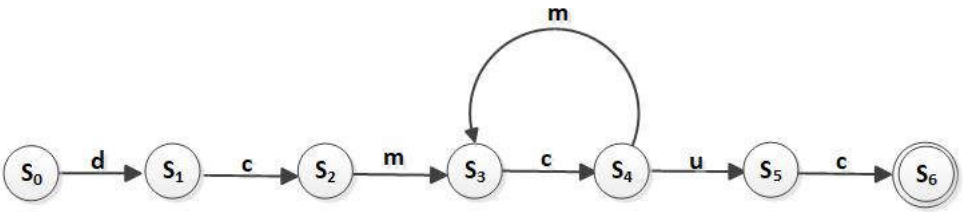


Fig.4 Scroll and Swipe timed automata
图 4 Scroll 和 Swipe 时间自动机

MultiTouch 事件的触发需要两根手指同时操作.为了识别 MultiTouch 事件,Minitouch 规定不同手指的 Press 事件通过字符串 d0 或 d1 标示、Move 事件通过字符串 m1 或 m2 标示、Release 事件通过字符串 u1 或 u2 标示.不同手指事件的字符串 d0、m0、u0 和 d1、m1、u1 可以连续被接收也可以在提交一个 c 字符串后交替被接收.根据以上定义,我们给出了 MultiTouch 事件的自动机,如下图 5(其中S_{23}代表 MultiTouch 事件识别成功)所示, AppCheck 将 MultiTouch 事件抽象为 Press, Move, Release 三个阶段,分别对应图 5 的左侧部分、中间部分以及右侧部分.自动机接收到字符串 d0 或 d1 后开启 MultiTouch 事件并进入 Press 阶段.根据上文所述的规则,字符串 d0 和 d1 可以连续被接收也可以在提交一个 c 字符串后交替被接收,因此从状态S_0到状态S_7共有四条路径.同理,如图 5 的中间部分和右侧部分所示,在 Move 阶段的状态 S_7到S_{15}以及 Release 阶段的状态 S_{15}到S_{23}之间同样也有四条路径.不同之处在于,因为 MultiTouch 事件包含一个以上的 Move 事件,所以当 S_{15}在接收到字符串 m0 或 m1 后,需回到状态S_8或S_9,继续停留在 Move 阶段以便继续接收新的 Move 事件字符串. 当状态 S_{15}接收到字符串 u1 或 u2 后,自动机将进入 Release 阶段,经过上文所提到四条路径后到达终止状态S_{23}.根据两个手指间的距离变大或变小,AppCheck 将 MultiTouch 事件进一步划分为：zoom-in 以及 zoom-out,分别代表多点触屏中放大手势以及缩小手势.

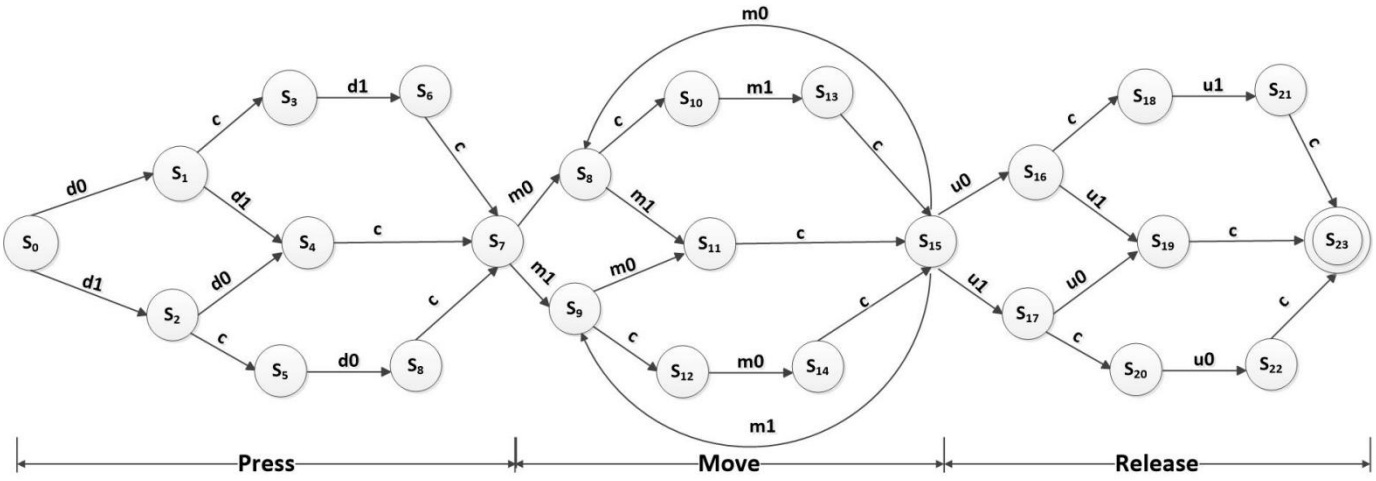


Fig.5 MultiTouch Event timed automata
图 5 MultiTouch 事件时间自动机

为了识别与以上操作序列的相关 UI 组件,AppCheck 通过遍历 Android UI 树直到所遍历组件坐标在事件触碰的屏幕范围内(当有多个组件符合条件时,根据组件的 text、clickable、editable 等属性进一步筛选组件),当成功遍历到满足条件的组件

后,AppCheck 会记录下组件的 ResourceID、index、class、type、text、coordinates、clickable、以及 Android UI 树中此叶节点的 XPath 等作为 selector 定位器.

selector 用于精确地标识与操作相关联的 UI 元素,并且独立于设备的屏幕大小和分辨率.为了唯一标识并查询到目标 UI 组件,AppCheck 定义并应用了三种定位方式：ResourceID 加 index 定位器、属性定位器和 XPath 定位器来识别与用户操作相关联的 UI 组件.Resource ID 加 index 定位器存储 UI 元素的 ResourceID 以及该 UI 元素的 index.属性定位器基于两个属性标识一个元素：元素的类和元素所显示的文本.XPath 定位器根据 UI 元素在 2.1 中提取 Android UI 树中的位置标识一个元素.我们的方法不使用 ResourceID 作它唯一的定位器.原因有两个：首先,在 Android 框架中不是每个资源元素都会有 ResourceID(有些 UI 元素的 ResourceID 为空);其次在一棵 Android UI 树中可能会有 ResourceID 重名的情况.这三种不同的定位方式,将应用在方法的重放阶段中用来唯一识别与用户操作序列相关联的 UI 组件.

2.3 跨设备重放

为了重放 2.2 中记录的操作序列,一种可行的方法是将操作序列所生成的测试用例转换成可以被现有 Android 平台测试框架执行的测试脚本(如 Android Espresso [8], Robotium [9]).然而,这种方法需要被测试设备连接到后端服务器后,再运行 adb 命令执行测试用例.很显然,这种方法不适用于众包测试的场景.

为支持在众包测试环境中完成平台无关的操作序列(2.2 中抽象出的操作序列)重放,我们开发了一个基于 Android 平台辅助功能[21]的重放应用程序.该应用程序主要完成三个任务：1)为测试用例搭建一个适当的运行环境.2)执行测试用例.3)将收集到的运行数据传送到 AppCheck 服务器端.接下来将从以上三点介绍 AppCheck 跨设备重放的具体实现：

运行环境搭建：启动该应用后,测试人员可以从 AppCheck 提供的列表中下载测试脚本和被测应用,重放应用程序会自动安装被测应用.安装完成后 AppCheck 将启动被测应用并执行测试脚本.

执行测试用例：AppCheck 解释器将逐行读取测试脚本,直到所有脚本语言执行完毕.当重放失败时,解释器会定位重放失败的语句并生成重放失败报告.

脚本语言的执行主要使用 Android Accessibility Service 所提供的 API 实现,通过使用 Accessibility Service 测试设备无需连接 adb 便可完成操作序列重放.例如,它可调用 findAccessibilityNodeInfosByID() API 通过 ResourceID 定位 UI 组件,也可以通过调用 findAccessibilityNodeInfosByText() 通过文本寻找 UI 组件,并根据 UI 组件的属性值来判断搜索到 UI 组件是否符合要求,如符合要求 AppCheck 将调用 performAction() API 去执行相关操作.如果上述方法无法找到符合要求的 UI 组件,AppCheck 会根据 XPath 定位器去遍历 UI 树直到发现与操作序列相关的唯一 UI 组件,然后触发相应的操作.

在重放过程中,可能会出现某个操作导致某个 Activity 或 View 被重新加载,但在新的 Activity 或 View 没有完全加载之前,下一个操作被调度执行的情况.为了避免这种情况,现有的大多数技术和工具(例如 Robotium [9]、Uiautomator [10])插入一些等待原语(例如,Waitfortext,Waitforactivity).然而,这些等待原语的插入需要特定领域知识.自动化完成这些等待原语的插入非常困难且容易出错.

为了解决这一问题,AppCheck 监听了由 Android 系统维护的系统级事件：State-changed 事件(State-changed 事件触发时,某个 Activity 或 View 会被加载).当 State-changed 事件被触发时,我们认为某个 Activity 或 View 将会被加载, AppCheck 将等待 2000 毫秒(Google 官方给定的执行时间标准)后调度执行下一个操作.

运行数据传送：在重放每个操作之前,AppCheck 将收集当前运行应用程序的 UI 结构信息以及应用截图.每个操作响应时间(响应时间的计算方法将在 2.4 中介绍)和系统日志也同时被记录下来.这些运行数据将在重放完成后被传送到 AppCheck 服务器端.

2.4 兼容性问题检测

根据从 2.3 收集到的数据,AppCheck 从功能、性能和显示三个方面检测应用跨设备重放的兼容性问题.接下来将从这三个方面介绍 AppCheck 兼容性检测过程的具体实现：

功能：功能兼容性问题检测分为异常识别和 Android UI 树匹配两个部分,分别负责检测异常信息和 UI 结构.

性能：为了检测跨设备重放时发生的性能问题,AppCheck 监听了由 Android 系统维护的系统级事件：Content-changed 事件.当 Content-changed 事件触发时,当前执行界面的内容会发生改变(例如,用户使用软键盘输入后 Activity 内某个 EditText 内容发生变化).通过监听 Content-changed 事件,我们对操作序列跨设备重放的响应时间给出了如下定义：

定义 2(AppCheck 操作跨设备重放响应时间)本文定义操作序列跨设备重放的响应时间为从操作被 AccessibilityService 调度执行后到新窗口第一个 Content-changed 事件发生后的时间间隔.

根据定义 2,在重放的每一个操作之前,AppCheck 会通过在调度执行 Click、LongClick、Input、Swipe、Scroll、MultiTouch 以及 EntityKey 等操作函数中嵌入 System.currentTimeMillis() API 完成对每个操作执行开始时间的记录,并监听此后第一个 Content-changed 事件的触发时间,并以此为依据计算出不同设备上每个操作的响应时间.这些运行数据将被收集并发送到后端服务器,用于检测兼容性问题.

显示：为了全面检测跨设备重放时发生的显示问题,我们将显示检测分为三个步骤：Structure check 检测、OCR(Optical Character Recognition,光学字符识别)检测、Color Histogram(颜色直方图)检测. AppCheck 兼容性问题检测内容列表见表 2：

Table 2 Compatibility issue detection range of AppCheck

表 2 Appcheck 兼容性问题检测范围

分类		检测内容
功能	异常识别	负责检测测试设备重放时是否存在异常.
	Android UI 树匹配	负责检测测试设备和参考设备用户界面是否包含相同的用户界面元素.
显示	Structure check 检测	负责检测测试设备和参考设备的用户界面元素布局是否一致.
	OCR 检测	负责检测测试设备和参考设备的界面元素实际显示文本是否一致.
	Color Histogram 检测	通过使用颜色直方图算法,检测测试设备和参考设备的用户界面实际显示颜色是否一致.
性能	日志分析	根据定义 2,检测操作序列中是否有操作超过 Google 官方推荐的执行时间标准.

根据上文所述的兼容性问题检测范围, AppCheck 将从三个方面、五个子过程检测兼容性问题,具体实现如算法 2 所示：

算法 2:兼容性问题检测算法 CID(Compatibility Issue Detection)

输入：RDModel：录制时收集到的参考设备运行状态信息集合,集合中每个元素包含 log(日志)、screenshot(截图)、tree(Android UI 树)等信息. TDModel：重放时收集到的测试设备运行状态信息,与 RDModel 数据结构相同.
输出：CIDReport 兼容性问题检测报告

```
01：Begin
02：  CIDReport = null
03:  //异常识别
04：  Foreach line in tdmodel.log
05:         If line.indexof("Exception") != null then
06:            CIDReport.add(Exception-CID(line))
07:         End If
08:    End For
09：    // Android UI 树检测
10：    //定义检测属性集合
11：   prop ← {"package","class","resource-id","selected","focused","index","content-desc","password","long-clickable",
12：              "scrollable","focusable","clickable","checkable"}
13：   For i = 0; i < RDModel.size; i++
14:      rdmodel = RDModel.get(i)
15:      For j = 0; j < TDModel.size(); j++
16:         tdmodel = TDModel.get(j)
17：         Foreach child element n1i in rdmodel.tree
18：            Foreach child element n2i in tdmodel.tree
19:               If n1i.name == n2i.name then
20:                  matchNode, matchProp ← 0
21:                  Foreach property p in prop
22:                        If n1i.p = n2i.p or (n1i.p.scrollable = true or n2i.p.scrollable = true) then
23:                           matchProp++
24:                        Else If n1i.p.scrollable = fasle and n2i.p.scrollable = fasle then
25:                           CIDReport.add(AndroidUITreeCheck-CID(n1i,n2i))
26:                        End If
27:                  End For
28:                  If matchProp = prop.size then
29:                        matchNode++
30:                  End If
31:               End If
32:            End For
33:         End For
34:         If matchNode = rdmodel.tree.size or then
35:            // Structure check 检测阶段
36:            bool ifStructMatch = StructureCheck (rdmodel, tdmodel, CIDReport)
37：            If ifStructMatch then
38：               //Ocr check 检测阶段
39：               bool ifOcrMatch = OcrCheck (rdmodel, tdmodel, rdmodel.screenshot, tdmodel.screenshot)
40:                If ifOcrMatch = false then
41:                     CIDReport.add(OcrCheck-CID(rdmodel, tdmodel, rdmodel.screenshot, tdmodel.screenshot))
42:                End If
43：                // Color Histogram 检测阶段
44:                bool ifColorHistogramMatch = ColorHistogramCheck(rdmodel.screenshot, tdmodel.screenshot)
```

```
45:                  If ifColorHistogramMatch = false then
46:                     CIDReport.add(ColorHistogramCheck-CID(rdmodel.screenshot, tdmodel.screenshot))
47:                     ImageCompare(rdmodel.screenshot, tdmodel.screenshot)
48:                  End If
49:                  If ifColorHistogramMatch and ifOcrMatch then
50:                     CIDReport.add(rdmodel +""+ tdmodel+" Pass")
51:                  End If
52:               Else
53:                  ImageCompare(rdmodel.screenshot, tdmodel.screenshot)
54:               End If
55:            End If
56:            //性能检测
57:            Foreach line in tdmodel.log
58:               If line.indexof("LoadTime") != null then
59:                  If LoadTime > 2000 or LoadTime < 500 then
60:                     CIDReport.add(LoadTime-CID(line))
61:                  End If
62:               End If
63:            End For
64:         End For // TDModel 元素遍历结束
65:      End For // RDModel 元素遍历结束
66:    return CIDReport
67: End
```

CID 算法的主要输入有：录制时收集到的参考设备运行状态信息 RDModel 集合,RDModel 集合中包括所有 Android UI 树的 log(日志)、screenshot(截图)、tree(Android UI 树)等信息.重放时收集到的测试设备运行状态信息 TDModel,其数据结构与 RDModel 相同.算法的输出是兼容性问题检测报告 CIDReport,算法具体执行过程如下：

算法初始化：首先将 CIDReport 赋值为空(第 2 行);

异常检测：算法将首先进行异常检测,通过逐行读取测试设备的日志来检测重放期间是否发生异常,如果检测到异常则在 CIDReport 集合中插入一条异常检测失败信息(第 3-8 行);

Android UI 树检测：异常识别完成后,CID 算法将需要检查的属性集合赋值给 prop(第 10-12 行),接着遍历 RDModel 和 TDModel 中所有截图、Android UI 树以及日志信息并取出参考设备每棵 UI 树中的每一个节点 $n1_i$,去匹配测试设备每棵 UI 树中的每一个节点 $n2_i$.首先判断每个根节点的子节点是否命名相同(第 13-19 行).如果相同,进一步抽取每个子节点的 prop 集合属性,对比这些属性的值是否相等.如果每个节点所匹配属性数量 matchProp 与 prop 集合的属性数量相等则继续进入 Structure check 检测阶段,如果不相等,算法将检查 $n1_i$ 和 $n1_i$ 的 scrollable 属性是否为真,如果为真说明当前控件加载不完整,未匹配控件可能会通过 Scroll 操作后匹配成功,无法确认是一个兼容性问题需要进一步检测.如果两个节点的 scrollable 属性都为假则在 CIDReport 集合中插入一条 Android UI 树兼容性检测失败信息(第 20-35 行).

结构检测：StructureCheck 函数负责检测测试设备和参考设备用户界面元素的布局是否正确(第 36 行), 算法 3 描述了具体的检测过程.

算法 3: 结构检测算法 SC(Structure Check)

输入: 不同 UI Tree 的两个结点 node1 node2
输出：兼容性问题检测报告 CIDReport

```
01：Begin
02：   If node1.parent = node2.parent then
03：      St ← getSiblingNodeSet(node1)
04：      Sr ← getSiblingNodeSet(node2)
05：      Foreach sibling Node n1_i in St
06：         At ← getAlignInfo(n1_i,node1)
07：      End For
08：      Foreach sibling Node n2_i in Sr
09：         Ar ← getAlignInfo(n1_i,node2)
10：      End For
11：      Foreach element a1_i in At
12：         Foreach element a2_i in Ar
13：            If a1_i not equeal a2_i then
14：               CIDReport.add(StructCheck-CID(a1_i ,a2_i))
15：               return false
16：            End If
17：         End For
18：      End For
```

```
19:       T ← getRelativePosition(node1, node1.parent)
20:       R ← getRelativePosition(node2, node2.parent)
21:       If [T.Origin - R.Origin] <σand [T.End – R.End] <σand [T.Origin – R.Origin] <σand [T.End – R.End] <σthen
22:          return true
23:       Else
24:          CIDReport.add(StructCheck-CID(T,R)
25:          return false
26:       End If
27:    End If
28: End
```

SC 算法的主要输入有：不同 UI 树的两个结点 node1、node2.算法的输出是兼容性问题检测报告 CIDReport.其中,算法第 1-10 行负责计算每个节点与父节点和兄弟节点的位置关系并生成结构布局图.第 11-18 行检测测试设备和参考设备相对应节点的布局关系是否一致.第 19-28 行检测测试设备和参考设备相对应节点在父结点内的相对位置是否一致. 具体执行过程如下.

1）结构布局图生成：SC 算法将需要进行检测的两个结点父结点进行对比,如果父节点相同则调用 API getSiblingNodeSet() 分别将两个结点的所有兄弟结点加入集合 St 和 Sr(第 1-4 行),接着通过使用 getAlignInfo()计算两个结点与其他兄弟节点的位置关系并将位置关系分别存储在集合 At 和 Ar 中(第 6-10 行)并转换为结构布局图.AppCheck 定义两个子节点的位置关系为{letf-top, top ,right-top,letf-bottom,bottom,right-bottom, overlapping,contain}八种.转换过程如下图 6 所示,左侧 AnkiDroid 截图的父节点是一个 linearlayout 布局控件,包含三个文本框控件, SC 算法会将左侧截图界面布局转换为右侧的结构布局图.

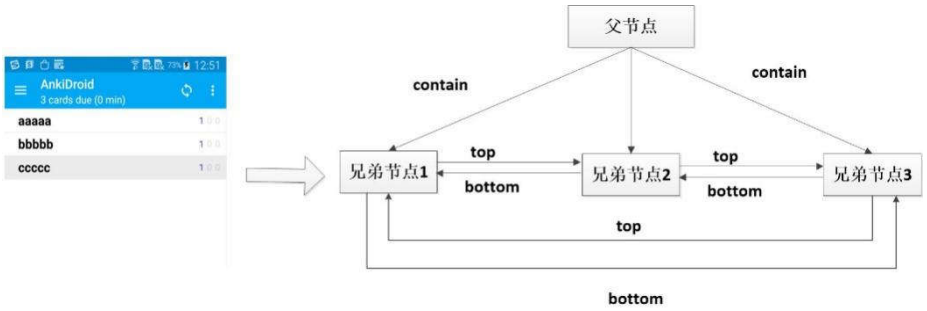


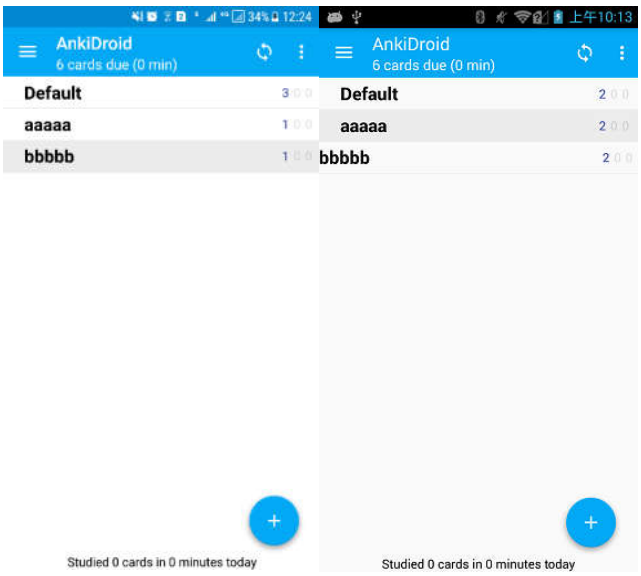
Fig.6 Structure Check example
图 6 Structure Check 示例

2）布局关系一致性检测：接着对比集合 At 和 Ar 结构布局图的位置关系是否相同,如果不同则说明存在兼容性问题 Structure Check 失败,算法将在 CIDReport 集合中插入一条 Structure Check 检测失败信息.如相同则进入相对位置一致性检测(第 11-18 行).

3）相对位置一致性检测：这一阶段将使用 getRelativePosition() API 计算两个结点在父结点控件内的相对位置,其中 T 和 R 代表测试设备和参考设备的界面元素结点,Origin 和 End 分别代表界面元素结点的起始坐标范围.如果两个结点在父结点内相对位置范围的差小于阈值 σ 则认为输入两个结点通过了 Structure check,反之则在 CIDReport 集合中插入一条 Structure Check 检测失败信息. (第 19-28 行).

OCR 检测：为了检测界面控件布局一致但文本显示不一致的情况.通过 Structure Check 检测后,算法将进入 OCR(Optical Character Recognition,光学字符识别)检测阶段. OCR 检测阶段通过使用 OcrCheck() API 识别出测试设备和参考设备界面元素实际显示的文本,并对比测试设备和参考设备实际显示的文本,如相同则通过 OCR 检测进入 Color Histogram 检测阶段,如不同将在 CIDReport 集合中插入一条 OCR 检测失败信息并使用 ImageCompare() API 进一步检测兼容性问题原因(第 38-42 行).

Color Histogram 检测：进入 Color Histogram 检测阶段后,AppCheck 将使用 ColorHistogramCheck() API 来检测测试设备和参考设备用户界面实际显示的颜色是否相同,如果检测结果小于阈值 σ 则可认为检测设备和参考设备的用户界面颜色一致并通过 Color Histogram 检测,如不同则使用 ImageCompare() API 进一步检测兼容性问题原因并在 CIDReport 集合中插入一条 Color Histogram 检测失败信息 (第 43-46 行).例如在测试应用 AnkiDroid v2.5alpha64 时(如图 7 所示),AppCheck 首先将从参考设备和测试设备中获得的截图和 Dump 文件分别转换为 Android UI 树和结构布局图(结构布局图转换方法见算法 3),接紧着进行 Android UI 树检测,当检测通过后继续完成布局检测、OCR 检测以及颜色直方图检测,如有任何检测失败则使用 ImageCompare() API 进一步检测兼容性问题原因(第 47 行).在本示例中三星 S7(Android 7.0)文本可以正常对齐(如图 7(a)所示),而在红米 1(Android 4.3)中存在文本不对齐的兼容性问题（如图 7(b)所示),在布局检测阶段中会被发现, AppCheck 将通过图片比较标出兼容性问题的具体位置,以供测试人员进一步分析兼容性问题原因. 通过 OCR 以及 Color Histogram 检测后,算法将认为此次重放通过了兼容性检查(第 43-51 行),继续进入性能检测.



(a) (b)

Fig.7 AnkiDroid compatibility problem detection example

图 7 AnkiDroid 兼容性问题检测例子

性能检测：通过 OCR 以及 Color Histogram 检测后,算法将进入性能检测阶段,本阶段将依据定义 2 检测重放是否存在性能问题.如果操作的响应时间小于 500 毫秒或者大于 2000 毫秒,算法将在 CIDReport 集合中插入一条性能检测失败信息(第 56-63 行).如果操作的响应时间符合推荐标准,AppCheck 将认为被测应用在测试设备上通过兼容性检测并返回兼容性问题检测报告 CIDReport.

完成以上检测后, CID 算法将分别取出 RDModel 和 TDMode 集合中的下一棵 Android UI 树重复以上检测过程,直到两个集合中所有 Android UI 树完成兼容性检测 (第 13-65 行).

3 实现

我们基于上文提到的方法实现了一种基于录制/重放的 Android 应用众包测试工具 AppCheck.它主要包括四个模块：事件序列收集器、用户操作抽象模块、跨设备重放模块以及跨设备兼容性问题检测模块.接下来介绍每个模块的实现.

事件序列收集器的实现基于开源框架 Open STF[20], AppCheck 利用其中的 minitouch[28]库记录各种触屏操作和 Menu,Home and Back 等实体键操作.同时为了快速提取应用界面信息,AppCheck 优化了 UIAutomator[30]的生成速度.用户行为的抽象模块是使用 Java 实现的,它通过根据定义 1 中的时间自动机来解析用户操作序列.跨设备的重放模块是一个 Android 移动应用.主要使用 Android Accessibility Service 所提供的 API 实现,通过使用 Accessibilityservice 测试设备无需连接 adb 便可完成操作序列重放.例如,它可调用 findAccessibilityNodeInfosByID() API 来通过 ResourceID 定位 UI 组件,也可以通过调用 findAccessibilityNodeInfosByText() 通过文本寻找 UI 组件,并根据 UI 组件的属性值来判断搜索到 UI 组件是否符合要求.为了实现基于 XPath 定位器搜索 UI 组件,该模块调用 getRootInActiveWindow() API 获取 Android UI 树的根元素,并继续调用 performAction() API 去执行相关操作.为了防止用户在重放期间的错误操作而导致重放失败,重放应用会自动运行一个透明窗口来防止用户的错误操作.重放应用在执行操作序列的同时也会收集测试应用各种数据(例如 UI 布局、应用截图),并发送给后台服务器,后台服务器使用 Node.js 实现.跨设备兼容性问题检测模块是用 Java 实现的,该模块是通过对比测试设备上和参考设备收集到的数据是否一致来检测兼容性问题.并最终通过 HTML5 提供的画布功能,以 HTML 网页形式报告检测结果.

兼容性检查中的 OCR 功能使用 Tesseract OCR [44]引擎实现,并使用 OpenCV 中的 Color Histogram 算法以及 Yahoo 的 Blink-Diff[45]分别完成相似度检查以及图片差异查找功能.

4 实验设计与结果分析

为了评估我们方法的有效性和效率,我们在八个不同的平台对 AppCheck 进行了测试,包括智能手机、Phablets、平板电脑和模拟器,对各大应用市场的 100 个 Android 应用进行了实验.在实验中,我们主要考虑以下研究问题：

问题一：Appcheck 是否可以支持跨设备记录/重放？如果可以,它与现有的技术相比效率如何？

问题二：Appcheck 跨设备操作序列收集和重放的性能开销如何？

问题三：Appcheck 是否可以有效发现应用程序的兼容性问题？

接下来,我们将详细介绍实验以及实验结果.

4.1 实验环境搭建

在实验中,我们选择了一组不同的 Android 设备,它们反映了 Android 平台上硬件和软件的多样性.所选设备的配置见表 3,其中三星 S5 被选为测试参考设备(其运行结果作为测试断言),其余设备被选为测试平台.屏幕尺寸范围从 4.3 到 10,包含所有主流移动设备：智能手机和平板电脑.后端服务器具有配置如下：8G 内存,英特尔(R)核心(TM)2 四核 2.67 GHz.

Table3 Target Android smart device list

表 3 目标 Android 智能设备列表

设备名	屏幕尺寸	分辨率	系统版本
Samsung S5	5.1	1920×1080	5.0
RedMi Note 4	5.5	1920×1080	6.0
Samsung I8268	4.3	800×480	4.3
HTC 10	5.2	2560×1440	7.0
Samsung GALAXY Tab	10.0	2048x1536	6.0
Smartisan T1	4.95	1920×1080	4.4.

为了探讨问题一,我们选取了 14 个应用程序从 F-Droid——个开源的 Android 应用程序库,和 86 个来自中国应用程序商店(如安智、小米等)的应用(见表 4).选定应用程序的涵盖范围广泛,如社交网络(如微信),新闻(如腾讯新闻)、阅读应用(如书旗小说)、流媒体应用(例如爱奇艺)、图像编辑应用(如美图秀秀)和邮箱(例如 K-9 邮件).为了收集实验数据进行评估,我们还招募了四名未参与研究的学生.当他们通过浏览器与测试应用交互时,AppCheck 将捕获发生的各种触摸事件.我们为每个选定应用程序收集 10 个以上不同的操作序列.这些操作序列将被抽象并在测试设备上重放.

Table 4 Top 100 Android Applications chosen from the main application markets

表 4 各大应用市场挑选出的 100 个移动应用

应用名	类别	版本	应用名	类别	版本	应用名	类别	版本
微信	社交聊天	v6.3.2	QQ 同步助手	实用工具	v6.8.8	墨迹天气	居家生活	v5.9.2
手机 QQ	社交聊天	v6.6.3	电信营业厅	实用工具	v6.4.0	美图秀秀	摄影摄像	v5.1.1.0
百度贴吧	社交聊天	v9.4.8	违章查询助手	实用工具	v5.8.0	天天 P 图	摄影摄像	v5.7.1
微博	社交聊天	v6.7.0	Dalvik Explorer	实用工具	v3.9	AnkiDroid	教育学习	v2.5alpha64
QQ 空间	社交聊天	v6.6.1.2	猎豹清理大师	实用工具	v6.04.1	小猿搜题	教育学习	v7.4.0
豆瓣	社交聊天	v5.26.0	Adobe AIR	实用工具	v30.0	网易有道词典	教育学习	v7.7.2
旺信	社交聊天	v4.5.7	Adobe Flash Player	实用工具	v11.1	金山词霸	教育学习	v8.3.7
阿里星球	社交聊天	v10.0.7	动态壁纸	实用工具	v1.1.2	驾校一点通	教育习	v6.0.55
陌陌	社交聊天	v8.7.9	BeeCount	实用工具	v2.3.0	QQ 阅读	图书阅读	v5.11.0.8
探探	社交聊天	v3.0.4	智联招聘	实用工具	v6.3.0	书旗小说	图书阅读	v10.6.9.67
知乎	社交聊天	v4.3.0	计算器	实用工具	v3.0	百度地图	旅行交通	v10.7.0
心悦俱乐部	社交聊天	v4.9.0	手机淘宝	时尚购物	v7.9	滴滴出行	旅行交通	v5.2.6
爱奇艺	影音视听	v7.3	天猫	时尚购物	v5.20.1	优步-Uber	旅行交通	v5.0.8
优酷	影音视听	v5.7	聚美优品	时尚购物	v3.887	摩拜单车	旅行交通	v7.5.0
爱奇艺 PPS	影音视听	v7.8.0	百度糯米	时尚购物	v8.3.0	途牛旅游	旅行交通	v9.49.0
暴风影音	影音视听	v7.5.8	京东商城	时尚购物	v7.0.6	携程旅行	旅行交通	v7.11.2
QQ 影音	影音视听	v3.2.0	小米商城	时尚购物	v4.2.9	去哪儿旅行	旅行交通	v8.3.8
土豆视频	影音视听	v5.8.4	美丽说	时尚购物	v9.5.1	手机百度	新闻资讯	v10.7.5
QQ 音乐	影音视听	v6.1.1	一号店	时尚购物	v6.1.3	腾讯新闻	新闻资讯	v5.6.00
乐视视频	影音视听	v7.15	美丽衣橱	时尚购物	v3.1.9	今日头条	新闻资讯	v6.7.5
搜狐视频	影音视听	v6.9.6	百度云	效率办公	v7.13.0	汽车之家	新闻资讯	v6.0.0
PPTV	影音视听	v7.2.3	Adobe Reader	效率办公	v16.1.1	天天快报	新闻资讯	v4.8.10
百度视频	影音视听	v8.4.1	网易邮箱	效率办公	v6.2.1	中国建设银行	金融理财	v4.0.7
央视影音	影音视听	v6.5.0	WPS	效率办公	v9.7.5	大智慧	金融理财	v8.62
百度音乐	影音视听	v5.8.0	K-9Mail	效率办公	v5.010	同花顺	金融理财	v8.70.41
虾米音乐	影音视听	v2.6.12	腾讯微云	效率办公	v6.5.3	光大银行	金融理财	v4.1.7
风行视频	影音视听	v3.2.2.4	美团	居家生活	v6.6.4	中国银行	金融理财	v1.5.25
支付宝	实用工具	v9.6	天气通	居家生活	v5.96	邮政银行	金融理财	v1.6.1
Firefox	实用工具	v47.0	12306	居家生活	v4.0.0	小米运动	体育运动	v3.3.6
搜狗输入法	实用工具	v8.20.1	饿了么	居家生活	v7.4.0	最右	娱乐消遣	v4.3.2
百度网盘	实用工具	v8.12.7	百度外卖	居家生活	v5.0.0			
UC 浏览器	实用工具	v12.0.2	大众点评	居家生活	v10.1.11			
ConnectBot	实用工具	v1.9.2	58 同城	居家生活	v8.12.3			
Boss 直聘	实用工具	v6.051	美团外卖	居家生活	v6.6.4			
QQ 安全中心	实用工具	v6.7.2	赶集网	居家生活	v7.4.1			

在这个实验中,我们将分别使用 AppCheck 和 Mosaic(另一个基于坐标等比例缩放以完成跨设备记录/重放的技术)来录制/重放以上 100 个应用,因为 Mosaic 没有开放源代码,我们根据 Mosaic 论文中提供的方法基于 RERAN 实现了 Mosaic.为了公平起见,我们要求学生进行两次同样的交互操作,并分别使用 AppCheck 和 Mosaic 完成操作序列的收集.为了提高重放成功率, Android 设

备从 AppCheck 服务器端下载屏幕尺寸相近设备生成的测试脚本,并且人工审核以保证重放环境与录制环境相同,审核内容包括关闭除测试应用以外的程序、网络带宽、测试脚本内容是否与测试用例相同、GPS 定位位置等.

为了回答研究问题二,我们根据实验发现系统开销主要在两个方面：首先,是由提取 App UI 元素布局信息引起的.其次是由于截图和网络通信造成的.为了评价 AppCheck 的系统开销,我们选择了 5 个流行的应用,并收集每个应用程序的 5 个操作序列.在红米 Note 4 记录每个操作序列的平均执行时间,每一个操作序列测试三次选取平均值.

为了回答研究问题三,我们使用 AppCheck 测试 8 个已知的兼容性问题.针对每个兼容性问题设计一个可以重现兼容性问题的测试脚本,以测试 AppCheck 是否可以正确检测到兼容性问题.

4.2 实验结果及分析

根据上节的实验设计,实验一步骤设计如下：

步骤 1：根据每个应用的主要功能设计测试用例.原则上,设计的用例应覆盖该应用与其他应用或系统交互的场景.

步骤 2：将收集的事件序列转换为可跨设备的中间脚本语言脚本.

步骤 3：在目标设备的完成重放.

重复步骤 1 到步骤 3 直到所有 100 个应用都测试完毕.

通过以上实验步骤,实验结果如下表 5 和表 6 所示：

Table 5 AppCheck successfully recorded and replay applications

表 5 AppCheck 成功录制/重放的应用列表

AppCheck 成功录制重放的应用列表	
类别	成功录制/重放的应用数量
聊天社交	13
影音视听	10
实用工具	15
时尚购物	11
效率办公	6
居家生活	5
摄影摄像	2
教育学习	5
图书阅读	2
旅行交通	6
新闻资讯	5
金融理财	5
体育运动	1
娱乐消遣	1

Table 6 Mosaic successfully recorded and replay applications

表 6 Mosaic 成功录制/重放的应用列表

Mosaic 成功录制重放的应用列表	
类别	成功录制/重放的应用数量
聊天社交	3
影音视听	5
实用工具	4
时尚购物	2
效率办公	4
居家生活	1
摄影摄像	1
教育学习	1
图书阅读	1
旅行交通	1
新闻资讯	1
金融理财	1
体育运动	0
娱乐消遣	0

实验结果表明,基于实验收集到的用户交互事件序列,AppCheck 可以成功录制重放各大应用市场挑选出的 100 个主流移动应用中 87 个应用* (这里成功录制重放指收集到的用户交互事件序列转换为平台无关测试脚本后,可以在表 3 的测试设备上正确执行,没有异常抛出),而 Mosaic 在大多数场景下录制/重放失败,只能在 25 个成比例缩放的场景下重放成功.经过分析 AppCheck 重放失败的 13 个应用我们发现重放失败原因有以下几点：

(1) 传感器数据：AppCheck 是基于 GUI 的测试方法,无法获取传感器数据,导致某些需要传感器数据的测试场景重放失败.

*该实验数据集分为四部分，访问地址如下：

1) https://gitee.com/TETTT/appcheck_test_data_set_part_i; 2) https://gitee.com/TETTT/appcheck_test_data_set_part_ii

3) https://gitee.com/TETTT/appcheck_test_data_set_part_iii; 4) https://gitee.com/TETTT/appcheck_test_data_set_part_vi

(2) 随机场景：某些银行类应用,因为安全方面的考虑密码按键位置每次会随机发生改变,导致录制/失败;

(3) Seletor 获取失败：WebView 等非 Android 原生界面元素的应用,无法通过辅助功能、HierarchyViewer、UIautomatorviewer 和 WindowManager 这四种方法获得界面信息;

(4) 操作不支持：目前 AppCheck 只支持 Click、LongClick、Input、Swipe、MultiTouch 以及 EntityKey 等操作的录制/重放,某些操作如三个手指的 MultiTouch、精确度高的拖拽操作无法支持;

(5) 不确定性：应用本身有不确定性(例如从网络上加载内容、非确定性函数的调用等)导致界面不一致;

这些不足可以通过插桩捕获传感器输入、录制过程中收集非确定性函数调用返回值等方法解决.我们计划在将来的工作中解决这些问题.

为了回答研究问题二,表 7 和表 8 分别展示了 AppCheck 在跨设备录制/重放时的额外系统开销以及 AppCheck 所支持操作的平均执行时间.通过分析表 7 可知相对于被测应用的日常使用场景,AppCheck 额外增加了系统开销 20%左右.这是因为为了保证录制脚本的跨设备重放,需要有额外网络通信和截图开销导致的.表 8 是通过本文定义 2 计算得出每个操作的平均执行时间,通过分析表 8 的实验数据可知 Click、LongClick、Input、Swipe、MultiTouch 以及 EntityKey 等操作的执行时间都低于 2000 毫秒,符合 Google 官方给定的执行时间标准.

Table7 Top 5 Apps Average Overhead

表 7 5 大流行应用平均时间开销

应用名称	时间开销			录制文件大小
	原始版本(s)	重放(s)	时间开销占比(%)	(KB)
微信	170	203	19.41	52
手机 QQ	201	240	20.89	72
爱奇艺	322	393	22.04	103
优酷	212	251	18.39	85
支付宝	297	362	21.88	100

Table 8 The average execution time of the operations supported by AppCheck

表 8 AppCheck 所支持操作的平均执行时间

操作名称	平均执行时间(ms)
Click	1017
LongClick	1646
Input	1103
Swipe	1150
MultiTouch	1223
Back	822
Home	723

为了回答研究问题三我们选取了 8 个已知的兼容性问题,其中包括 6 个功能相关的兼容性 Bug 以及 2 个性能相关的兼容性 Bug.针对每个兼容性问题设计一个可以重现 Bug 的测试脚本以及相对应的测试环境,以测试 AppCheck 是否可以正确检测到问题,实验结果见表 9：

Table9 Detection of compatibility problems

表 9 兼容性问题检测

应用名称	版本号	引用链接	Bug 描述	Bug 类别	是否重现
AnkiDroid	v2.5alpha64	https://github.com/ankidroid/Anki-Android/commit/4f3909c	AnkiDroid v2.5alpha64 在 Android5.0 以上系统中文本框文本不对齐	显示	Y
AnkiDroid	v2.2alpha21	https://github.com/ankidroid/Anki-Android/commit/ddce379	AnkiDroid v2.2alpha21 在 Android4.3 系统中文本显示不全	显示	Y
支付宝	v9.6	http://www.miui.com/forum.php?mod=viewthread&tid=4250687&highlight	支付宝 v9.6 在 MIUI9.5 系统中不能正常登录	功能	Y
12306	v2.0	http://www.miui.com/forum.php?mod=viewthread&tid=4250687&highligh	12306 v2.0 在 Android5.0 以上版本系统中启动时闪退	功能	Y
12306	v2.0	http://www.miui.com/forum.php?mod=viewthread&tid=4250687&highligh	12306 v2.0 在 MIUI7.2 系统中启动时卡顿无任何响应	性能	Y
今日头条	v5.0	http://www.miui.com/thread-6411642-1-1.html	今日头条 v5.0 在 MIUI8 系统中卡顿	性能	N
微信	v6.12.15	http://www.miui.com/thread-6687709-1-1.html	微信 v6.6 在 MIUI8 系统中不能使用微信转账和发红包	功能	N
K-9 Mail	v5.010	https://github.com/k9mail/k-9/issues/1461	K-9 Mail v5.010 在 Android6.0 系统中无法在 SD 卡上保存附件	功能	Y

通过分析表 9 中的结果,可知 AppCheck 可以有效检测出 8 个兼容性问题中的 6 个.经过分析 AppCheck 未检测出的 2 个兼容性问题,我们发现检测失败的原因有以下几点：

(1) 图片识别算法局限性：当前图片识别算法在分析分辨率不同的截图时存在局限性,导致兼容性问题误报.

(2) 应用后台修复：当软件公司收到用户反馈后,会推送修复补丁,应用会在在用户无法察觉的情况下后台修复,导致兼容性问题检测失败.

(3) 传感器数据：AppCheck 是基于 GUI 的测试方法,无法获取传感器数据,这导致无法检测与传感器相关的兼容性问题.

(4) 不确定性：应用本身有不确定性(例如从网络上加载内容、非确定性函数的调用等)导致界面不一致.

针对以上问题,我们计划将在今后工作中改进我们的检测算法.

5 局限性

为了克服当前众包测试人工操作会产生诸如输入错误、误操作等问题,AppCheck 引入了录制/重放技术,以确保重放可以正确完成,同时 AppCheck 使用了 Android 平台的辅助功能接口去完成重放操作,使用辅助功能不仅具有无需 root 设备的优点,还不需要同后台服务器建立 adb 连接(现有的测试框架都需要使用 USB 接口建立 adb 连接以完成自动化测试).然而辅助功能 MultiTouch 重放功能只能在最新的版本中可使用(例如,MultiTouch 的重放仅在 Android API 级别 24 以上可用).我们正在研究以解决目前方法的局限性.

AppCheck 另外一个局限性是无法检测到某些 Hybrid App 的 WebView 界面元素,这是因为 UIAutomator 无法识别 WebView 中的控件元素.但这个问题可以通过引入 Web 代理解决,具体方法可参考我们的另一个工作 X-Check [32] .目前 Hybrid App 正慢慢成为主流,我们将在未来的工作中扩展 AppCheck 以支持 Hybrid App 界面元素识别.

目前,AppCheck 主要关注跨 Android 设备完成触屏操作的录制/重放.同时,因为 AppCheck 忽略传感器的输入数据(例如,网络信息、GPS)和一些非确定性函数(例如,Random(),Date()),会导致误报一些兼容性问题.但这些不足可以通过插桩、捕获传感器输入和录制过程中收集非确定性函数调用返回值解决.我们计划在将来的工作中解决这些问题.

6 相关工作

6.1 Android自动化测试

当前,工业界和学术界提出了众多方法以支持智能手机测试用例的录制/重放.例如 Android SDK 工具包中的 Monkey 工具[42]可以向应用程序发送随机事件流,但这限制了 Bug 检测的效率.测试框架如 Robotium[9]和 Android Espresso[8]支持脚本生成和事件序列执行,但是需要手工编写测试脚本.

在 Android 应用自动化测试领域,基于 GUI 和模型以自动生成测试用例的方法非常流行.AndroidRipper[41]通过用户界面接口使用深度优先搜索以生成测试用例.A3E[39]则包含了两种搜索策略：深度优先和目标优先.SwiftHand [15]通过动态建立 GUI 的有限状态机模型,寻求以最少事件数达到最高的测试覆盖率.ORBIT[40]基于动态 GUI 抓取和静态代码分析方法,以避免生成不相关的 UI 事件.然而最近实证研究表明[7],所有现有技术的覆盖率都很低(小于 50%).

6.2 Android Record/Replay 框架

目前,工业界和学术界已有多种针对 Android 应用交互行为录制/重放的框架.工业界相关研究如 TestIn[16],以在不同型号的真机上进行测试为主,通过将应用运行在不同分辨率、不同尺寸和其他硬件配置的真机上完成对应用适配性的测试.在学术界,以 RERAN[12]为代表的记录底层日志方法,通过记录 Android 底层接口得到系统操作日志流,以完成测试用例的重放,其精确度可以达到毫秒级.但单纯记录操作日志流方法只能在录制的智能设备上重放,无法实现跨设备重放.Mosaic[13] 是 RERAN[12]工作的进一步扩展,通过将移动设备的指令集进行抽象化处理,将设备事件抽象为 Press, Move, Release 等有限的几个动作,来描述用户的所有行为,以达到跨设备重放用户行为的目的,其系统开销仅为 0.2%,并能跨设备和平台重放 Google Play 中的 45 个应用.但 Mosaic 存在两方面的局限性：(1)在指令集抽象方面,将移动设备复杂的指令集抽象为简单几个动作,以这种方式生成测试用例不具备可读性,无法重用.(2)在跨设备自适应方面,由于 Mosaic 采用成比例缩放方式实现不同屏幕尺寸和分辨率的适配,当应用界面布局随着屏幕旋转或屏幕尺寸、分辨率发生变化后,该方法会失效.

以 VALERA[46]为代表的插桩方法虽然可以在多个智能设备上重放,但这种方法只能测试单一应用,当前多个应用交互的场景越来越普及,单纯针对单个应用测试已经无法满足自动化测试需求.同时对字节码插桩会造成系统额外开销,不能准确的评估应用运行情况;另外对应用重签名会破坏应用自身的安全机制,如对支付宝、在线商城等应用程序重签名,会对应用程序自身的安全机制造成破坏.

SPAG[37]是基于录制/重放技术的 Android 应用测试工具,它通过事件批处理和智能等待等功能以减少重放过程中的不确定性,并整合 Sikuli IDE 以在每个 GUI 操作后通过断言(截图图片)验证应用是否正确执行.SPAG-C[38]是 SPAG 工作的进一步扩展,其目

的是增加测试断言的可重用性而不影响测试精度.它从外部摄像机捕获程序截图,以进一步减少记录测试用例所需时间,并提高测试断言的可重用性而不影响准确性.

与以上录制重放框架不同,AppCheck 在录制阶段支持众包方式收集用户交互事件序列并转换成平台无关的测试脚本,在重放阶段借助于 Android 辅助功能服务支持直接在众包用户的设备上进行执行测试用例.

6.3 兼容性检测

当前针对 Android 生态系统的碎片化问题已有一些工作进行研究.Dan Han 等人研究了 HTC 和摩托罗拉在 Android 问题跟踪系统中的错误报告,指出 Android 生态系统是零散、缺乏可移植性的[33].为了更好地理解 Android 应用程序中碎片诱导的兼容性问题,工作[35]对 191 个开源软件的兼容性问题进行了实证研究,总结了一些常见模式来检测 Android 应用程序中的兼容性问题,并设计实现了 FicFinder. Mona Erfani 等提出 CHECKCAMP[34]自动化测试技术针对 iOS 和 Android 平台的原生应用.CHECKCAMP 通过抽象每个平台的执行序列,通过比较不同平台的执行序列来检测兼容性问题.X-Checker[36]是一种跨平台的应用程序开发框架(例如 Xamarin),它开发了一种差异测试技术,以识别这些框架在源平台和目标平台 API 的不一致性.

与以上兼容性检测方法相比 AppCheck 通过收集测试用例在不同设备上执行时生成的各种运行时相关数据(例如,截图和布局信息)以检测兼容性问题,支持检测 Android 应用在不同设备上运行时产生的行为、布局以及性能等方面的一致性问题.

7 结论

本文提出了一种基于录制/重放的 Android 应用众包测试方法,该方法一个重要特点是可以通过互联网以众包测试方式支持测试用例的生成和执行.基于该方法我们实现了一种基于录制/重放的 Android 应用众包测试工具 AppCheck.通过我们的初步实验结果表明,AppCheck 可以有效实现众包测试环境下 Android 应用用户交互行为的跨设备录制/重放,并在重放完成后识别兼容性问题,改进了当前方法的不足.

致谢 在此感谢中科院软件研究所软件工程技术研究开发中心的老师们以及北京城市学院的王溯、徐自强、李林清、汪伟光同学对本文实验的大力协助.

References:

[1] 18796 Different Android Devices According to OpenSignals Latest Fragmentation Report. [Online].Available: https://thenextweb.com/mobile/2014/08/21/18796-different-android-devices-according-opensignals-latest-fragmentation-report/.

[2] An eye-opening report from OpenSignal found that there are at least 24,093 distinct Android devices in the wild. , [Online]. Available:http://www.techspot.com/news/61666-android-fragmentation-least-24093-distinct-devices-wild.html.

[3] Google I/O 2017 By the Numbers: 2 Billion Android Devices, 500 Million Photos Users, and More [Online].Available: http://gadgets.ndtv.com/apps/news/google-io-2017-android-drive-photos-users-youtube-maps-1695234.

[4] W. Choi, G. Necula, and K. Sen. Guided GUI Testing of Android Apps with Minimal Restart and Approximate Learning. In OOPSLA, 2013. 623–640.

[5] A. Machiry, R. Tahiliani, and M. Naik. Dynodroid: An Input Generation System for Android Apps. In FSE, 2013.

[6] R. Mahmood, N. Mirzaei, and S. Malek. EvoDroid: Segmented Evolutionary Testing of Android Apps. In FSE, 2014.

[7] S. R. Choudhary, A. Gorla, and A. Orso. Automated test input generation for Android: Are we there yet? In ASE'15, pages 429–440.

[8] Google Espresso. [Online].Available:https://developer.android.com/training/testing/espresso/

[9] Robotium. [Online].Available:https://github.com/RobotiumTech/robotium

[10] Goolge UI Auotomator. [Online].https://developer.android.com/training/testing/ui-automator.

[11] Appium. [Online].Available:http://appium.io/

[12] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein. RERAN: Timing- and Touch-sensitive Record and Replay for Android. In ICSE 2013.

[13] M. H. Y. Zhu, et al. Mosaic: Cross-Platform User-Interaction Record and Replay for the Fragmented Android Ecosystem. In ISPASS 2015.

[14] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon. Using GUI Ripping for Automated Testing of Android Applications. In ASE 2012.

[15] W. Choi, G. Necula, and K. Sen. Guided GUI Testing of Android Apps with Minimal Restart and Approximate Learning. In OOPSLA, 2013.

[16] Testin. http://testin.cn/

[17] Utest. https://www.utest.com/

[18] Crowdsurced testing. https://en.wikipedia.org/wiki/Crowdsourced _testing.

[19] Wang, Junjie, et al. Towards effectively test report classification to assist crowdsourced testing. ASE, 2016

[20] Smartphone Test Farm (STF). https://openstf.io/.

[21] Google Accessibility.[Online].Available:https://www.google.com/accessibility/.

[22] Didi Chuxing. [Online].Available:https://www.didiglobal.com/

[23] JD App. [Online].Available: https://app.jd.com/

[24] JD App Bug. [Online].Available:http://www.miui.com/thread-4545053-1-1.html
[25] L. Wei, Y. Liu, et al. Taming Android fragmentation: characterizing and detecting compatibility issues for Android apps. In ASE 2016.S
[26] S. Roy Choudhary, M. R. Prasad, and A. Orso. X-pert: Accurate iden- tification of cross-browser issues in web applications. In ICSE, 2013.
[27] Wang, Junjie, et al. Towards effectively test report classification to assist crowdsourced testing. ASE, 2016
[28] Minitouch. [Online].Available:https://github.com/openstf/minitouch.
[29] Hierarchy-viewer. https://developer.android.com/studio/profile/hierarchy-viewer.
[30] UIAutomator [Online].Available:http://developer.android.com/tools/testing/testing_ui.html.
[31] Firebase. https://firebase.google.com/docs/test-lab/
[32] G. Wu, M. He, et al. Detect cross-browser issues for JavaScript-based Web applications based on record/replay. ICSME 2016.
[33] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia. Understanding Android Fragmentation with Topic Analysis of Vendor-Specific Bugs. In WCRE 2012.
[34] M.E Joorabchi, M. li. A. Mebah. Detect Inconsistencies in Multi-Platform Mobile Apps. In ISSRE, 2015.
[35] L. Wei, Y. Liu, et al. Taming Android fragmentation: characterizing and detecting compatibility issues for Android apps. In ASE 2016.S
[36] Boushehrinejadmoradi, N., et al. Testing cross-platform mobile app development framework. In ASE 2015.
[37] Y.D. Lin, E.T.H. Chu, et al. Improving Accuracy of Auotmated GUI Testing for Embedded systems. IEEE software, 2014.
[38] Y.D. Lin, J.F. Rojas, et al. On the Accuracy, Efficiency, and Reusability of Automated Test Oracles for Android Devices. IEEE TSE, No.10 2014.
[39] T.Azim, I. Neamtiu.Targeted and Depth-first Exploration for Systematic Testing of Android Apps. In OOPSLA 2013.
[40] W. Yang, M.R. Prasad, T. Xie. A Grey-box approach for automated GUI Model generation of Mobile applications. In FASE 2013
[41] D. Amalfitano, S.D. Carmine, A. Memon, et al. Using GUI Ripping for automated Testing of Android Applications. In ASE 2012.
[42] Application/UI Exerciser Monkey. https://developer.android.com/studio/test/monkey.html
[43] Yu Kang,Yangfan Zhou,Min Gao,Yixia Sun,Michael R. LyuExperience Report: Detecting Poor-Responsive UI in Android Applications In 2016 IEEE 27th International Symposium on Software Reliability Engineering
[44] Tesseract-OCR. https://github.com/tesseract-ocr/tesseract
[45] blink-diff. https://github.com/yahoo/blink-diff
[46] Y. Hu and I. Neamtiu. Valera: an effective and efficient record-and- replay tool for Android. In Proceedings of the International Workshop on Mobile Software Engineering and Systems, pages 285–286. ACM, 2016.

附中文参考文献:

[47] 陈军成,薛云志,赵琛.一种基于事件处理函数的 GUI 测试方法.软件学报,2013,24(12):2830-2842.