安卓应用 HTTP 缓存缺陷的动态检测*

杨嘉成[12+]，彭鑫[12]，赵文耘[12]

1. 复旦大学 软件学院, 上海 201203
2. 上海市数据科学重点实验室, 上海 201203

Dynamic Detection of HTTP Cache Deficiencies of Android Apps*

YANG Jiacheng[12+], Peng Xin[12], Zhao Wenyun[12]

1. Software School, Fudan University, Shanghai 201203, China
2. Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China
+ Corresponding author: E-mail: jiachengyang15@fudan.edu.cn

Abstract: Both users and developers of Android application are concerned with data traffic consumption and response time of an application. To solve these problems, many mobile applications make use of cache (especially HTTP cache) to effectively reduce traffic and energy consumption, and improve responsiveness. However, the proper use of cache in a mobile application is related to the application's features and specific functions. The improper use of cache in many applications not only would be ineffective, but also waste precious storage space in mobile device. For this purpose, this paper proposed a dynamic analysis based automatic detection method of HTTP cache deficiencies of Android apps, which can dynamically monitor interior cache behavior of apps, and analyze whether there are deficiencies, and what kinds of deficiencies. This paper realized a corresponding runtime library and an automated program transformation tool of Android application, and conducted an experiment on 19 present Android applications. The results show that the method is applicable for a large part of applications, and can effectively find problems like lack of cache, unlimited cache space and redundant cache in Android applications.

Key words: Android applications; HTTP cache; program transformation

摘　要: 移动应用数据流量消耗和响应速度是用户和开发者都非常关心的问题。为此，很多移动应用都使用了缓存技术（特别是 HTTP 缓存）以有效减少移动数据流量和电量消耗，加快应用响应速度。然而，移动应用缓存的合理使用与应用的特点和具体功能相关。许多应用中对于缓存的不合理使用非但不能起到应

*This work was supported by the High-Tech Research and Development Program of China under Grant No.2015AA01A203 (国家高技术研究发展计划), National Natural Science Foundation of China under Grant No. 61370079 (国家自然科学资金).
Received 2000-00, Accepted 2000-00.

有的作用，反而会占用移动设备宝贵的存储空间。为此，本文提出了一种基于动态分析的安卓应用缓存缺陷自动检测方法，可以在应用运行时动态监控应用内部的缓存行为，并分析其缓存是否存在缺陷，以及存在怎样的缺陷。本文实现了相应的运行时库以及安卓应用自动化改造的工具，并针对当前安卓应用市场上的 25 个安卓应用进行了检测。实验结果表明，该方法适用于安卓市场上的很大一部分应用，并能够有效发现安卓应用中存在的缓存缺失、缓存空间无限增长和缓存冗余等问题。

关键词： 安卓应用; HTTP 缓存; 程序改造

1 引言

网络通信能力在为移动应用带来强大支持的同时，也给移动用户带来了一些潜在的问题。首先，移动应用访问网络的过程会消耗数据流量，造成了用户的经济负担。其次，移动设备电池电量有限，而网络通信是移动应用中消耗电量最多的操作之一。研究表明，移动应用在非空闲状态下超过 40% 的电量消耗来自网络通信，其中由 HTTP（超文本传输协议，HyperText Transfer Protocol）操作所导致的电量消耗占近 80%[1]。最后，移动应用在通过网络获取数据并进行本地处理的过程中，网络传输可能由于网络状况不佳或数据传输量大而成为应用响应的瓶颈，影响用户体验。

缓存作为一种经典的技术手段，可以在移动设备端有效地缓解上述问题。安卓应用中的网络访问通常通过调用 HTTP 客户端 API（应用编程接口，Application Programming Interface）实现，其中常见的 HTTP 客户端包括 HttpURLConnection、Apache HTTP Client 和 OkHttp 等。但在这些 HTTP 客户端中，有些客户端本身缺乏缓存机制（如 Apache HTTP Client），其他一些客户端（如 OkHttp）虽然有内部的缓存支持，但仍存在以下问题：1）缓存需要开发者在代码中显式的启用，一些开发者由于疏忽而没有启用；2）HTTP 客户端 API 中的缓存处理抽象层次较高，无法针对特定应用需求进行优化处理，导致缓存效果不佳[2]。

由于上述原因，在实际开发中应用开发者往往需要自行定义并实现自己的缓存处理策略，这可能导致两方面的问题：1）未能充分利用缓存来优化应用性能；2）过度使用缓存或不及时清理缓存内容。后者会导致应用缓存占用过多的存储空间，以及缓存管理消耗较多的电量[3]，因此也会影响移动设备和应用的整体性能。

当前已经有一些研究工作关注于移动应用的缓存问题，但这些工作主要关注于移动端网络浏览器的 HTTP 缓存表现[4-6]。有些研究针对应用的 HTTP 缓存问题进行检测[7]，但策略比较简单且没有考虑安卓应用的特点，因此无法发现其中所包含的深层问题，比如应用缓存占用的设备空间、应用缓存的有效性等。为了实现细粒度的安卓应用缓存缺陷检测，发现与特定安卓应用特点相关的 HTTP 缓存问题，本文提出并实现了一种基于动态分析的安卓应用 HTTP 缓存缺陷检测方法。该方法可以针对安卓应用的 APK（安卓应用安装包）文件进行插装，使其具有在运行时监测并记录与 HTTP 缓存相关的行为和数据的能力。另一方面，该方法通过安卓应用自动化测试工具对插装后的应用进行自动运行同时收集运行日志，并基于日志分析发现应用中的 HTTP 缓存缺陷并定位所在位置。

为了验证该方法的有效性，我们利用基于该方法所实现的分析工具对当前安卓应用市场上的 25 个安卓应用进行了自动化分析。分析结果表明，所提出的方法适用于当前应用市场上的大多数应用。同时，该方法能够有效发现安卓应用中存在的缓存缺失、缓存空间无限增长和缓存冗余等问题。针对所发现的这些问题，本文还分析了出现这些问题的原因以及解决的思路。

2 背景

安卓应用通常是用 Java 语言编写的，然后打包成 APK 文件以供安装使用。从应用源代码到打包好的 APK 文件，一般要经过以下几个阶段的转换：

首先，应用源代码被编译成 Java 类文件；然后这些类文件被安卓 SDK（软件开发工具包，Software Development Kit）中的 dx 工具转换为.dex 文件，这种文件是可以运行在安卓的 Dalvik 虚拟机上的可执行字节码；最后，这些.dex 文件与声明文件、相关的资源文件（如图片、用户界面布局文件）一起被打包成 APK 文件，添加签名后就可以发布使用了。

安卓官方提供了两组 HTTP 客户端 API：HttpURLConnection 和 Apache HTTP Client。其中，前者是目前官方建议开发者使用的 HTTP 客户端 API；后者在安卓 API 22 级之后已经废弃，但由于一些较早开发的代码中仍有使用，出于兼容性的考虑，安卓对这部分代码保留了支持。此外，还有一些第三方 HTTP 客户端，如 OkHttp，可供安卓开发者使用，这些 HTTP 客户端一般也都是基于 HttpURLConnection 实现的。借助上述 HTTP 客户端 API，开发者可以非常方便的编写代码，获取 HTTP 资源。但值得注意的是，这些 HTTP 客户端本身要么缺少缓存机制接口供开发者使用，要么虽然提供了缓存支持，但可能需要开发者自行开启，而且缓存处理较为简单。

安卓设备中包含两个存储文件的区域：内部存储和外部存储。内部存储是设备内部提供的非易失性内存，外部存储一般是移动存储介质，如 SD 卡（安全数码卡，Secure Digital Memory Card）。通常来讲，安卓设备的内部存储在应用运行时是始终可用的，各应用的存储空间相互隔离，应用无法访问其他应用在内部存储中保存的文件；在用户卸载某个应用后，系统将移除该应用所保存的文件。而安卓设备的外部存储并不始终可用，在外部存储中保存的文件是全局可读的；用户卸载某个应用时，只有保存在某些特定路径（通过 Context 类中的 getExternalFilesDir 方法获得）下的文件才会被系统删除，应用保存在其他路径下的文件并不会被删除。安卓系统不会定期清理内部存储和外部存储中缓存文件夹下的文件，只有在内存不足时才会删除内部存储中的缓存文件。

3 方法和实现

3.1 方法

本文提出了一种基于动态分析的安卓应用缓存缺陷自动检测方法，该方法的概览如图 1 所示。

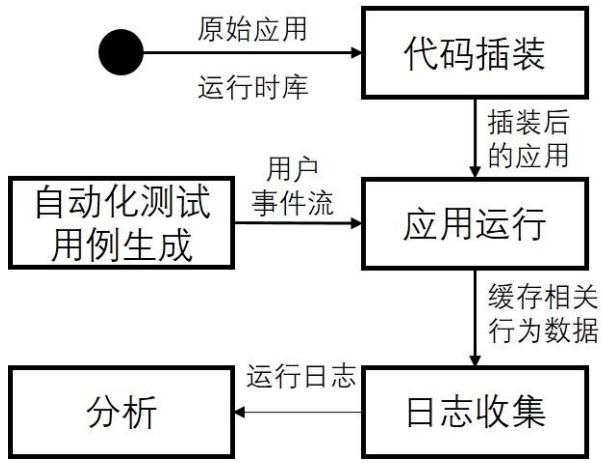

Fig.1 Overview of Android App HTTP Cache Detection

图 1 安卓应用 HTTP 缓存检测概览

首先，方法对安卓应用的 APK 文件进行插装改造。这一过程会向原始安卓应用中引入一个运行时库，生成的插装后的应用具有在运行时监测并记录与 HTTP 缓存相关的行为和数据的能力。这些数据包括：1）HTTP 响应头部信息（主要是缓存相关的信息，如失效时间、缓存控制和上次被修改时间）及具体内容；2）访问本地文件的绝对路径；3）文件访问时间；4）缓存文件容量；5）缓存文件访问；6）被删除文件的绝对路径。

改造过程中引入原始应用中的运行时库负责在应用运行时获取上述数据，该运行时库主要有三大职责：1）记录应用的文件访问和删除操作的相关信息；2）监测应用中的 HTTP 访问，记录 HTTP 响应的头部信息（统一资源定位符、过期时间、缓存控制和上次修改时间）以及响应的具体内容后，再将 HTTP 响应内容返回给应用中原本的调用，从而不会对应用的正常运行产生影响；3）应用启动时的运行时库初始化处理，包括设置应用包名、从文件系统中恢复该应用在之前运行时获得的数据

记录等，以及在应用退出时将内存中的数据记录持久化到设备文件系统中。

接着，改造后的安卓应用被安装到安卓设备上以供运行。在改造过的应用被启动后，方法借助安卓自动化测试工具（如 Monkey、Robotium 等），自动化的向安卓设备发送用户事件流。用户事件包括点击事件、触屏事件、手势操作和按键操作等。通过自动生成的用户事件流，模拟用户使用应用的过程，产生原始应用逻辑中访问文件、获取网络资源的操作。

在应用运行时，被引入的运行时库会实时监测并收集前述的与 HTTP 缓存相关的行为和数据，进行一定的处理整合后，生成运行日志。日志内容包括：1）应用中的 HTTP 请求的次数；2）导致不完善缓存发生的 HTTP 请求在代码中的位置；3）每个缓存文件的访问次数以及缓存文件总访问次数；4）应用占用的设备内部存储和外部存储空间的大小变化；5）应用运行过程中删除的文件的文件名，以及这些文件在删除前的容量。

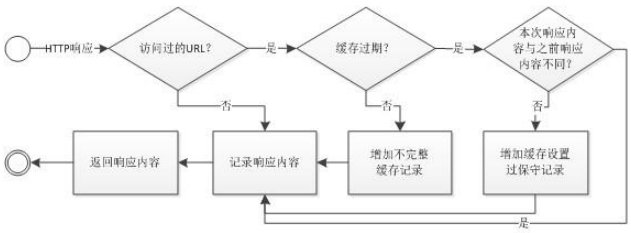

Fig.2 Interception and Analyze Process of HTTP Response

图 2 HTTP 响应拦截分析过程

运行时库通过在应用中监测 HTTP 响应及其内容来分析获得上述日志中的第 1、2 部分内容。图 2 展示了运行时库对 HTTP 响应进行拦截并分析其内容的过程。分析逻辑共分五个步骤：1）当在应用中发生一次 HTTP 访问时，运行时库会根据这次访问的网络资源的 URL（统一资源定位符，Uniform Resource Locator）查询在此之前应用是否访问过相同 URL 上的内容；2）如果访问过，那么分析程序会接着查找数据记录中最近的对应于该 URL 的缓存控制信息和缓存过期时间信息，否则结束；3）如果缓存并未过期，那么说明应用的 HTTP 缓存不完整，运行时库会将这次 HTTP 通信在程序代码中的位置记录到日志中；4）如果缓存过期，运行时库会查找获取数据记录中最近的对应于该 URL 的缓存内容，并将获得的缓存内容与此次 HTTP 访问获得的内容进行对比；5）如果对比发现两者内容并无差别，那么说明存在缓存控制设置过于保守的问题，运行时库会将这次 HTTP 通信在程序代码中的位置记录到日志中，否则直接结束。整个过程不会干扰应用获取 HTTP 响应的内容，因此也不会影响应用的正常运行。

最后，方法对运行获得的日志记录进行分析。分析的内容主要包括：1）应用缓存的完整性（也就是说，是否存在从网络中请求之前访问过并且还未过期的资源）；2）应用缓存控制设置是否保守；3）应用的缓存对设备内部和外部存储的空间占用状况；4）冗余缓存文件（未被使用过的缓存文件）数量，以及这些文件占所有缓存文件的比例；5）上次被访问时间在某一时刻之前的缓存文件的数量，以及这些文件占所有缓存文件的比例；6）缓存文件平均访问次数；7）应用对缓存空间占用的控制。对于 1、2 两部分内容，运行时库在运行时已经在日志中进行了记录；对其他五部分内容，分析程序将分别读入分析所需的运行日志中内容，然后按要求输出分析结果。需要注意的是，很多应用为了程序健壮性的需要，会在运行时生成一些临时文件和备份文件（如.bkp 文件）。分析程序会判断被删除文件的类型，剔除备份文件对分析结果的影响。

3.2 实现

本文实现了一个自动化的安卓应用代码插装工具，该工具的输入是 APK 文件，文件中包含的 Dalvik 字节码保留了应用的程序结构。插装过程中，工具会自动对 Dalvik 字节码进行分析和重写，引入我们实现的运行时库，使应用具有 3.1 节中描述的运行时 HTTP 缓存缺陷动态监测的能力。

图 3 是应用程序改造的概览。图中白色部分代表原本的安卓应用代码，灰色部分是本文的自动化

改造工具所引入的代码。dex2jar 是一个可以实现安卓中的.dex 文件和 Java 中的.class 之间的相互转换的工具。本文实现的工具借助 dex2jar，对 APK 文件中的 Dalvik 字节码进行转换后，对类文件进行操作，引入运行时库。对每一个 activity 类，改造工具会向其生命周期函数（onCreate,onResume 和 onStop 方法）中引入额外的代码，以在进入应用时对缓存监控进行初始化，并在离开应用时结束缓存监控，更新未持久化的数据记录。这部分改造的方法与 CollaDroid[8]中介绍的程序改造方法类似。

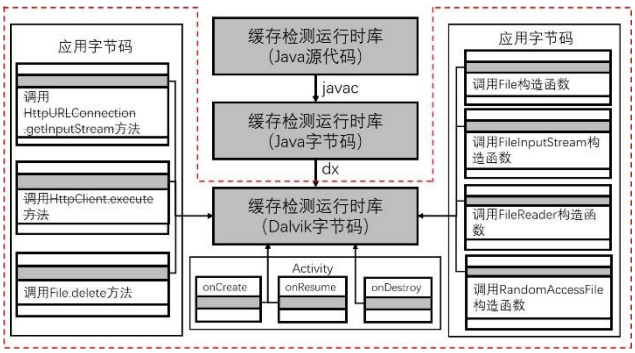

Fig.3 Overview of Program Transformation

图 3 应用程序改造概览

工具随后对应用所有类文件中每个方法的每一条语句进行遍历，查找与文件操作相关的语句和与 HTTP 资源访问相关的语句。

与文件操作相关的语句包括那些调用 Java 中的 File 类的构造函数、FileInputStream 类的构造函数、FileReader 类的构造函数、RandomAccessFile 类的构造函数,以及 File 类中的 delete 方法的语句。本文工具会在使用上述构造函数的语句之前引入额外的代码，调用运行时库中的方法，实现对缓存文件状态的监控。

```
1  public void someMethod(HttpURLConnection conn){
2      // 原始程序代码…
3      InputStream stream = conn.getInputStream();
4      // 引入的方法调用
5      InputStream stream =
6            Interceptor.interceptHttpURLConnection(conn);
7      // …
```

Fig.4 An Example of Program Transformation

图 4 程序改造示例

与 HTTP 资源访问相关的语句包括那些调用 java.net.HttpURLConnection 类中的 getInputStream 方法和调用 org.apache.http.client.HttpClient 接口中的 execute 方法的语句。调用上述两个方法，程序会获取网络中特定 URL 上的资源。图 4 展示了一个对调用 getInputStream 方法的代码进行改造的例子。请注意，这里为了方便理解，我们展示的是源代码，应用程序改造实际上是在 Dalvik 字节码上进行的。改造工具将使用运行时库中的方法来替换原始应用中相关的方法调用，从而在不影响应用正常使用的情况下实现对应用中发生的 HTTP 访问的监测。

4 实验和分析

4.1 实验设置

本文实现的 APK 改造工具是通过 Java 语言开发的，可以运行在任何安装了 Java 运行时环境的机器上。我们使用这一工具对原始的 APK 文件进行改造，然后安装在安卓设备上，并使用安卓 SDK 提供的 Monkey 测试工具模拟应用运行以获取数据。实验中用到的安卓设备包括一台三星 Galaxy S4（Exynos 5410 CPU，Imagination PowerVR SGX544 MP3 GPU，2GB RAM，16GB 内存）和一台红米 Note 4（联发科 Helio X20，Mali-T880 MP4 GPU，3GB RAM，64GB 内存）。

4.2 适用性

我们从安卓市场上下载了 25 款下载量较高，并且网络通信比较频繁的应用（如新闻客户端、在线音乐客户端）的 APK 文件，并使用本文所实现的自动化改造工具对这些 APK 文件进行自动化改造。实验发现，改造单个 APK 文件的耗时一般不超过 30 秒，并且改造过程本身都可以顺利完成，生成可以安装到安卓设备上的可用 APK 文件。改造后的 APK 文件中.dex 文件的大小比改造前增加了 0.4%到 2.1%，平均增加了 1.3%。由于 APK 文件中还会包含许多资源文件（如图片、布局文件等），.dex 文件容量一般只占 APK 中所有文件总容量的不到一半，因此改造对 APK 文件大小的影响

是微不足道的。

随后我们在设备上运行这些改造过的应用，发现其中有 6 款应用无法正常运行。这些无法正常运行的应用通常表现为闪退、进入应用后长时间停留在初始界面没有响应、应用提示当前应用不是官方版本后强制退出等。通过分析控制台日志信息等方式，我们认为这些应用之所以无法运行，是因为应用开发者在应用内设置了一些自我保护机制，使得被第三方修改过的应用无法正常运行。

实验所使用的应用类型多种多样，包括新闻、音乐、购物等等，与一些关注于移动浏览器应用缓存检测的工作[4]-[6], [11]相比，本文所提出的方法具有更广泛的适用性。

在改造过程中，我们发现有一些应用使用了 Apache HTTP 客户端来访问 HTTP 资源，目前的安卓官方开发文档中已经建议开发者不要再使用这种方式。对应用所使用的具体 HTTP 客户端的检测是先前的缓存检测工作[7]所无法做到的，这也是本文方法的优势之一。根据方法调用所在的类所处的包的名称，我们发现其中大多数调用是在应用使用的一些第三方库中，只有 2 款应用是在自己开发的代码中使用了 Apache HTTP 客户端。

实验结果表明，本文实现的工具可以对现成的安卓应用进行自动化的改造，并且这种改造适用于安卓市场上的很大一部分应用。

4.3 缓存检测结果及分析

我们对改造成功并可以在安卓设备上正常运行的 19 款应用进行了运行检测。为了自动化的生成用户事件流，我们使用了安卓 SDK 提供的 Monkey 工具，该工具可以向安卓模拟器或真实设备中发送伪随机用户事件流，模拟用户的触屏、点击、手势和按键等操作。借助 Monkey 提供的命令，我们将每段用户事件流中的用户事件数量控制在 500 个，用户事件包含触屏、位移、按键事件，事件将只发生在当前应用中，不会跳出到其他应用。为了给应用一定的响应时间，我们将用户事件输入间隔设置为 1 秒。对每个改造后的应用，我们每天分别运行两次，这两次运行中间的时间间隔大于 5 小时，共运行三天（为保证结果的可靠性，实验开始时应用都是全新安装的），得到的数据及分析结果如图 5 所示。

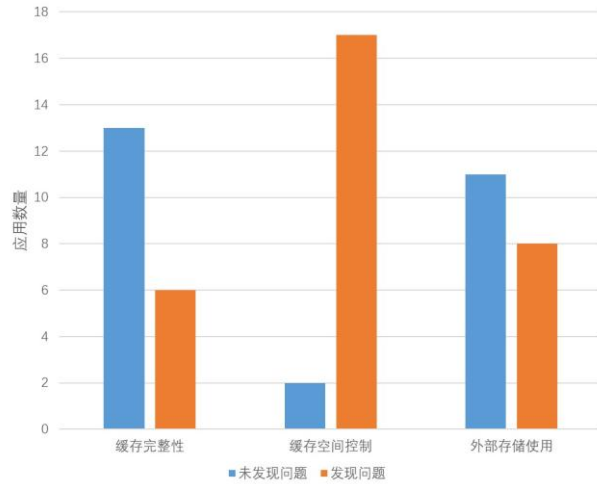

Fig.5 Result of Android App HTTP Cache Detection
图 5 安卓应用 HTTP 缓存检测结果

在我们测试的 19 款应用中，有 6 款应用的 HTTP 缓存不完整，存在从网络中重复获取相同未过期资源的情况，这意味着数据流量和电池电量的不必要消耗；同时，有 7 款应用存在 HTTP 头缓存控制设置过于保守的情况，也就是说，虽然某些资源根据 HTTP 头中的缓存控制信息已经过期，但实际上这些资源内容并未发生变化，这主要源于服务器端设置的过于保守。需要注意的是，当应用发生一次 HTTP 请求时，如果运行时库发现 HTTP 缓存不完整，就不会再检查缓存设置是否保守，因此实际当中缓存设置保守的问题可能要比实验结果中展示的要严重一些。Yifan Zhang 等人的工作[7]中所使用的应用 HTTP 缓存检测方法是在一个较短的间隔内点击两次应用的某个界面元素，比较两次点击产生的 HTTP 响应内容。这种方法只能简单的检测出应用的缓存是否完整，但没有考虑响应内容的过期时间、内容变化等因素，也就无法像本文所提出的方法一样检测更多更深入的应用缓存问题，如缓存控制设置、缓存的有效性等。

图 5 中还展示了应用对安卓设备内部存储、外

部存储的使用情况，以及对缓存总容量的控制。从图中可以看到，有 8 款应用并没有使用设备的外部存储，这些应用中不乏新闻客户端、在线音乐应用。虽然安卓官方文档中并没有明确提出开发者对设备内、外部存储的使用建议，但通常来讲，安卓设备的内部存储空间较外部存储空间更小，完全将缓存文件存储在内部设备上，特别是当缓存文件中可能包含比较多的图片、音频等容量比较大的资源时，可能给设备内部存储空间带来比较大的压力，造成在设备外部存储还剩余比较多空间时，内部存储空间已经告急的情况。同时，在实验检测的共 19 款应用中，我们发现只有 2 款应用对缓存文件的存储空间占用进行了控制，绝大多数应用的缓存空间在实验过程中都是无限制增长的。实验结束后应用产生的缓存文件大小一般在几十兆甚至上百兆，虽然有可能存在应用中缓存空间控制的阈限较高的情况，但考虑到目前测试设备的容量和移动设备上安装的应用数量（2016 年，美国每台移动设备平均安装约 89 个应用[9]），我们认为我们实验得到的结果还是具有说服力的。

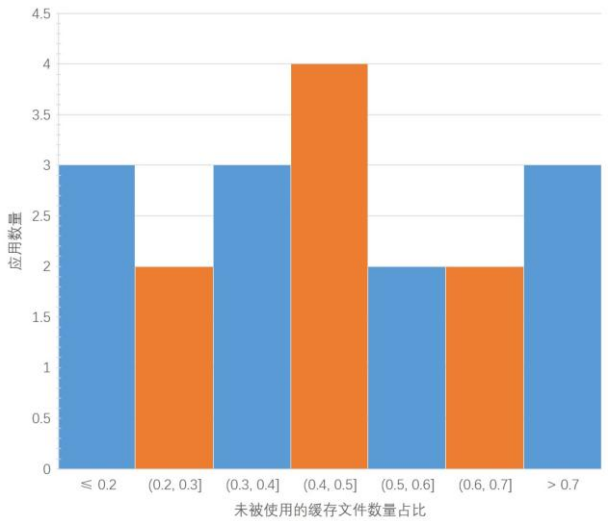

Fig.6 Proportion of Unused Cache Files
图 6 应用未使用的缓存文件数量占比

我们还对应用中未被使用的缓存文件数量占比进行了统计，统计结果如图 6 所示。从图中可以看出，有超过一半的应用，它们的缓存文件中有超过 40%是未被使用的。由于我们的实验进行了三天，每天运行每个应用两次，因此这些未被使用的缓存文件在未来被应用使用的几率也比较小，可以认为，这些缓存内容实际上对应用来说已经是冗余的了，基本不会给应用的性能带来什么实质性的帮助，反而造成设备存储空间的浪费。

在实验检测的 19 款应用中，有 16 款应用的缓存命中数与缓存文件数的比率在 3 以内，也就是说，平均每个缓存文件被访问的次数不到 3 次，由于其中大部分缓存文件没有被访问或只被访问过一次，这意味着缓存的使用集中在很小一部分文件中。

此外，应用中还有一部分缓存文件是长时间未被再次访问的，我们将“长时间”定义为应用最近两次运行（因为实验设置的运行间隔已经比较长）。请注意，这部分缓存文件是被使用过的。结合未被使用的缓存文件数量，在实验结束后，这两类缓存文件数量占缓存文件总数量的比例一般在 60%以上，如果去掉实验中应用最后两次运行过程中新增加的缓存文件，这一比例会更高。这一结果表明，应用中绝大部分缓存文件并不需要长期保留，应用完全可以清理这类缓存，而不会对应用性能造成太大的影响。

与现有的一些安卓应用 HTTP 缓存检测工作[4]-[7], [11]相比，本文所提出的方法不仅可以检测应用“有没有缓存”，而且可以通过监控应用对缓存文件的使用和管理状况，检测应用的缓存“好不好”，即有没有真正起到缓存应有的效果，有没有因为缓存而影响了设备的整体使用。特别是第二点，是现有的相关工作所忽视或是无法做到的。

5 讨论

第 4 章的实验结果表明，在目前安卓市场上的应用中，应用的 HTTP 缓存使用存在如下问题：1）部分应用的 HTTP 缓存并不完整，会重复获取网络中相同的资源内容，造成移动数据流量的浪费；同时，更多的网络请求意味着更多的电量消耗，在移动设备电池电量仍比较紧张的今天，这也是移动设

备使用者不愿意看到；2）一些应用虽然对运行过程中获取的 HTTP 资源进行了缓存，但往往忽略了对缓存的管理，没有对缓存文件的空间占用进行控制，导致缓存空间无限制增长，影响设备的整体使用；3）应用运行过程中产生的缓存文件中，有很大一部分是冗余的、无意义的，它们在生成后并没有被应用所使用，无法起到开发者期望的效果；同时，一些缓存文件虽然被使用过，但在应用随后的运行过程中不再被访问，这些缓存文件实际上也已经是冗余的了。

基于以上分析，针对安卓应用 HTTP 缓存，我们向开发者提出以下几点建议：

(1) 开发应用时应注意开启 HTTP 客户端 API 中提供的缓存，这是最基本的缓存处理，对开发者的编程负担很小，而且比较有效。

(2) 开发者应该根据所开发应用的特点及具体情况，设计实现更为合适、灵活的缓存处理策略，以获得更好的缓存效果。例如对于在线音乐客户端，用户常听的歌曲很适合缓存在本地，可以在很大程度上减少运行时消耗的流量，但用户使用类似“随便听听”的功能随机播放在线曲库中的歌曲时，缓存这类歌曲的对于减少流量消耗的意义并不大，因为用户可能不会重复听这些歌曲。

(3) 如果应用运行时使用的 HTTP 资源也是由应用的开发者部署在服务器上的，那么开发者应该谨慎对待资源缓存控制的设置，一方面要保证客户端不会展示过期的内容，另一方面又要把失效时间设置的尽可能久，以减少不必要的重复访问。

(4) 应用在运行时应该及时清理那些长时间未使用的缓存内容，减少缓存对设备存储空间的占用。虽然现在很多应用内置了供用户使用的清理缓存功能，但由于一台移动设备上应用数量往往有几十个甚至上百个，手动清理缓存对用户来说仍是一件比较繁琐的事情。应用应当把存储空间优先分配给那些使用频率较高的缓存内容，及时清理那些使用频率低或者长时间没有被使用的缓存内容，这样既可以保证应用的流畅运行，发挥缓存应有的效果，又可以保证良好的用户使用体验。

6 局限和未来工作

本文提出的方法及具体实现还存在一些局限，主要有以下几个方面：

(1) 由于不同的应用有各自的内部处理逻辑，我们对于缓存“命中”定义的粒度比较粗，只能定义到文件级别，因此应用存储缓存的方式和访问方式都可能会对分析结果产生影响。例如，应用如果将缓存集中存储在少数几个文件当中，那么这几个文件被访问的次数可能会比较多。此外，在应用运行时，可能会有一些保存在内存中的缓存内容，这些缓存内容的具体形式因应用而异，我们无法记录关于这类缓存的确切信息，如果应用不访问文件，而是直接获取存储在内存当中的缓存，那么可能会对检测的结果造成影响。不过由于设备内存容量相对设备总体存储容量比较有限，这类缓存的总容量一般比较小，对检测结果的影响相对有限。未来可以加入对应用代码结构的分析，找出应用存储 HTTP 缓存的数据结构和存储方式，以便在运行时获得更加准确的检测结果。

(2) 本文实验中的自动化测试工具使用的是安卓 SDK 提供的 Monkey，该工具生成的用户事件流的随机性比较大，可控性相对较小；而在一个移动应用中，不同的业务逻辑的使用频率也不尽相同，比如在微博客户端中，用户可能更常刷新微博而不是发送私信，Monkey 产生的随机用户事件流可能不能很好的模拟这一点，并可能对最终检测结果造成一定的影响。在未来的工作中，我们将尝试结合一些能产生更加可控的用户事件流的自动化测试工具来进行检测。

(3) 应用内容的更新频率和应用的使用频率也可能对检测结果造成影响。如果应用中的内容的更新频率很高（比如一些新闻客户端）而使用频率较低，检测得到的缓存命中次数可能相对较少。在本文实验中，我们每天检测两次，两次检测时间间隔大于 5 小时，检测三天，可以大致模拟普通用户使用应用的一般频率，但由于我们无法完全了解每个

应用的内容更新频率，因此实验的结果可能受到一些影响。

7 相关工作

有许多与安卓应用的 HTTP 缓存相关的工作。Yifan Zhang 等人[7]对安卓应用的网络缓存状况进行了调研，并且实现了一个系统级的应用网络缓存服务。这一工作进行的调研是通过代理服务器拦截应用的网络请求来实现的，只能实现对应用网络缓存的完整性的检测，无法对本文所描述的更深层次的、与应用相关的缓存问题进行检测。为了优化移动应用中的 HTTP 请求调用，减少 HTTP 请求对电池电量的消耗，Ding Li 等人[10]提出了一种通过静态分析自动检测安卓应用中的 HTTP 请求并在运行时将多个请求捆绑发送的方法，可以有效减少电量消耗，并且不会引入过多额外开销。

还有一些与移动端网络浏览器的缓存相关的研究工作。其中一部分工作[12,13]将移动设备与个人电脑上的网络访问行为的差异进行了比较，指出了这些差异可能对移动端的网络浏览器缓存产生的影响。还有一些工作[4,11]对移动端网络浏览器的缓存表现进行了研究和分析，虽然这些研究针对的是浏览器应用，但其中发现的诸如冗余传输、误缓存等问题在一些其他类型的移动应用中也同样存在。Xuanzhe Liu 等人[5,6]针对这些问题设计实现了一个基于代理的缓存优化系统，供移动端网络浏览器和服务器使用，可以节约网页加载时间，减少网络数据传输量。本文在上述基础上更进一步的研究了一般化的移动应用中存在的缓存问题，以及应用缓存的使用效率和空间占用情况。

一些现有工作对缓存策略进行了研究。Sunghwan Ihm 等人[14]研究发现，基于内容的缓存策略相比基于对象的缓存策略，可以获得更高的缓存命中率。Hyojun Kim 等人[15]对现有缓存策略在移动设备闪存上的适用性进行了调查研究，并针对移动设备闪存容量较小、性能有限的特点，设计了一种考虑了闪存中写操作的空间邻接性的缓存策略。Hariram Chavan 等人[16]提出了一种新的移动设备数据库缓存替换策略，该策略考虑了客户端运动的时间和空间属性，以及获取数据的模式。在移动设备上，该策略的缓存表现相较传统的最近最少使用策略有了较大提升。

8 总结

本文提出了一种基于动态分析的安卓应用缓存缺陷自动检测方法，该方法可以在运行时监测应用的 HTTP 通信和对缓存文件的使用，从而检测分析应用中存在的 HTTP 缓存问题。依据这一方法，我们实现了一个运行时库和一款安卓应用自动化改造工具，并使用 25 款安卓市场上的应用进行了实验。实验结果表明，本文提出的方法适用于很大一部分安卓应用，并能够有效发现安卓应用中存在的缓存缺失、缓存空间无限增长和缓存冗余等问题。本文还对发现的缓存问题进行了讨论，为开发者指出了解决这些问题的思路和方法。

References

[1] Ding Li, Shuai Hao, Jiaping Gui, William G. J. Halfond. An Empirical Study of the Energy Consumption of Android Applications[C]// Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution, Washington, DC, USA: IEEE Computer Society, 2014: 121-130.

[2] Shaiful Alam Chowdhury, Varun Sapra, Abram Hindle. Is HTTP/2 More Energy Efficient Than HTTP/1.1 for Mobile Users?[EB]. PeerJ PrePrints, 2015[2017-07-04]. https://peerj.com/preprints/1280/.

[3] Jing Li, Anirudh Badam, Ranveer Chandra. On the Energy Overhead of Mobile Storage Systems[C]//Proceedings of the 12th USENIX conference on File and Storage Technologies, Santa Clara, CA, Feb 17-20, 2014. Berkeley, CA, USA: USENIX Association, 2014, 105-118.

[4] Xuanzhe Liu, Yun Ma, Yunxin Liu, Tao Xie, Gang Huang. Demystifying the Imperfect Client-Side Cache Performance of Mobile Web Browsing[J]. IEEE Transactions on Mobile Computing: 2016, 15(9): 2206–2220.

[5] Xuanzhe Liu, Yun Ma, Yunxin Liu, Xinyang Wang, Tao Xie, Gang Huang. SWAROVsky: Optimizing Resource Loading for Mobile Web Browsing[J]. IEEE Transactions on Mobile Computing, 2016[2017-07-03].

http://ieeexplore.ieee.org/document/7801009/.

[6] Xuanzhe Liu, Yun Ma, Shuailiang Dong, Yunxin Liu, Tao Xie, Gang Huang. ReWAP: Reducing Redundant Transfers for Mobile Web Browsing via App-Specific Resource Packaging[J]. IEEE Transactions on Mobile Computing, 2016[2017-07-03]. http://ieeexplore.ieee.org/abstract/document/7762888/.

[7] Yifan Zhang, Chiu Tan, Qun Li. CacheKeeper: A System-Wide Web Caching Service for Smartphones[C]//Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing, Zurich, Switzerland, Sep 8-12, 2013. New York, NY, USA: ACM, 2013: 265-274.

[8] Jiahuan Zheng, Xin Peng, Jiacheng Yang, Huaqian Cai, Gang Huang, Ying Zhang and Wenyun Zhao. CollaDroid: Automatic Augmentation of Android Application with Lightweight Interactive Collaboration[C]// Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing, Portland, Oregon, USA, Feb 25-Mar 01, 2017. New York, NY, USA: ACM, 2017: 2462-2474.

[9] The Statistics Portal. Average Number of Apps Used vs. Number of Apps Installed by Users in The United States in 2016, by Device[EB]. Statistics, 2017[2017-07-03]. https://www.statista.com/app.php/statistics/681206/us-app-installation-usage-device/.

[10] Ding Li, Yingjun Lyu, Jiaping Gui, William G. J. Halfond. Automated Energy Optimization of HTTP Requests for Mobile Applications[C]// Proceedings of the 38th International Conference on Software Engineering, Austin, Texas, May 14-22, 2016. New York, NY, USA: ACM, 2016: 249-260.

[11] Feng Qian, Kee Shen Quah, Junxian Huang, Jeffrey Erman. Web Caching on Smartphones: Ideal vs. Reality[C]// Proceedings of the 10th international conference on Mobile systems, applications, and services, Low Wood Bay, Lake District, UK, June 25-29, 2012. New York, NY, USA: ACM, 2012, 127-140.

[12] Ioannis Papapanagiotou, Erich Nahum, Vasileios Pappas. Smartphones vs. Laptops: Comparing Web Browsing Behavior and the Implications for Caching[C]//Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems, London, England, UK, June 11-15, 2012. New York, NY, USA: ACM, 2012, 423-424.

[13] Chad Tossell, Philip Kortum, Ahmad Rahmati, Clayton Shepard, Lin Zhong. Characterizing Web Use on Smartphones[C]//Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Austin, Texas, USA, May 05-10, 2012. New York, NY, USA: ACM, 2012, 2769-2778.

[14] Sunghwan Ihm, Vivek S. Pai. Towards Understanding Modern Web Traffic[C]//Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference, Berlin, Germany, Nov 2-4, 2011. New York, NY, USA: ACM, 2011, 295-312.

[15] Hyojun Kim, Moonkyung Ryu, Umakishore Ramachandran. What is a Good Buffer Cache Replacement Scheme for Mobile Flash Storage?[C]//Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems, London, England, UK, June 11-15, 2012. New York, NY, USA: ACM, 2012: 235-246.

[16] Hariram Chavan, Suneeta Sane. Mobile Database Cache Replacement Policies: LRU and PPRRP[C]//Advances in Computer Science and Information Technology, Bangalore, India, Jan 2-4, 2011. Berlin, Germany: Springer, 2011, 131: 523-531.

YANG Jiacheng was born in 1993. He is a M.S. candidate at Fudan University. His research interests is mobile computing.

杨嘉成（1993–），男，山西太原人，复旦大学硕士研究生，主要研究领域为移动计算。

PENG Xin was born in 1979. He received the Ph.D. degree in computer science from Fudan University in 2006. Now he is a professor at Fudan University, and the senior member of CCF. His research interests in-clude requirements engineering, self-adaptive software systems, software product line, software maintenance and reverse engineering.

彭鑫（1979—），男，湖北黄冈人，2006 年于复旦大学计算机科学专业获得博士学位，现为复旦大学教授，CCF 高级会员，主要研究领域为需求工程，自适应软件，软件产品线，软件维护，逆向工程。

ZHAO Wenyun was born in 1964. He received the M.S. degree from Fudan University in 1989. Now he is a professor at Fudan University, and the senior member of CCF. His research interests include software reuse, software product line and software engineering.

赵文耘（1964—），男，江苏常熟人，1989 年于复旦大学获得硕士学位，现为复旦大学教授，CCF 高级会员，主要研究领域为软件复用，软件产品线，软件工程。