ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2000/00(00)-0000-00
doi: 10.3778/j.issn.1673-9418.

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-89056056

MultiOffload：一个支持多种资源借调的安卓应用 Offloading 平台*

张晨曦[1], 武翔宇[2], 许畅[1]

1. 计算机软件新技术国家重点实验室 南京大学计算机科学与技术系，南京 210023

2. 微软（中国）有限公司苏州分公司，苏州 215000

MultiOffload: An Android Application Offloading Platform Supporting Offloading Multiple Resources*

Chenxi ZHANG[1], Xiangyu WU[2], Chang XU[1]

1. Department of Computer Science and Technology & State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
2. Microsoft China Co. Ltd., Suzhou 215000, China
+ Corresponding author: Chang XU E-mail: changxu@nju.edu.cn

Chenxi ZHANG, Xiangyu WU, Chang XU. MultiOffload: An Android Application Offloading Platform Supporting Offloading Multiple Resources. Journal of Frontiers of Computer Science and Technology, 2000, 0(0): 1-000.

Abstract: The number of Android applications is increasing rapidly. However, some Android applications running on the smartphones are limited by the conditions of the smartphones themselves, such as insufficiency of CPU resources or malfunctioned sensors. They also face the problem of remote control due to their richer application scenarios. Computation offloading is proposed to solve this problem by transferring the complex computation tasks contained in the app to a server to execute and then returning the result back to the app. But there still remain problems such as complicated client-server connecting. This paper presents MultiOffload, an Android application offloading platform which supports Computation-offloading, Sensing-offloading and Touch-offloading at the same time. The smartphones which have already installed MultiOffload can find and connect each other to form a P2P network. The applications on the platform can borrow computation, sensor and screen resources from other devices. We extend existing

*This work is supported by the National High Technology Research and Development Program of China (863 Program) under Grant No.2015AA01A203，国家高技术研究发展计划（863 课题 2015AA01A203）；the National Natural Science Foundation of China under Grant No.61472174 and No.61690204，国家自然科学基金（面上项目 61472174、重大课题 61690204）

Computation- and Sensing-offloading approaches and enhance them with the support for Touch-offloading via program instrumentation. At last we give a demo to confirm the feasibility of our offloading platform design and choose two apps to display the offloading effect.
Key words: Android Application; Computation Offloading; Sensing Offloading; Touch Offloading

摘　要：安卓应用数量飞速增长，应用场景不断丰富，但部分安卓应用的运行受到智能手机本身 CPU 性能不足或传感器功能缺失的限制；同时在使用场景上也面临如何远程控制的问题。将应用内部的部分计算任务转移到远程服务器中计算是一种普遍的解决手机性能不足的方法，但是该方法仍存在服务器连接复杂等方面的问题。本文提出了 MultiOffload，一个同时支持安卓应用计算任务卸载与转移执行、传感器功能“借用”和触摸控制转移的平台，搭载这个平台的移动设备可以互相发现并建立 P2P 网络，平台上的移动应用能够“借用”移动设备计算资源、传感器资源和屏幕资源；重新实现和完善已有的计算任务卸载、传感器借用的插桩技术并提出触控转移的插桩实现方案。最后实现了一个示例以说明平台设计的可行性，并选取两个安卓应用插桩展示资源借用的效果。

关键词：　安卓应用；计算卸载；传感器借用；触控转移
文献标志码：A　　中图分类号：TP311

1 引言

信息时代，移动应用的数量呈现爆发式的增长。截止到 2017 年 5 月，在 Google Play 上发布的 Android 应用数量已经从 2012 年的 60 万增长到现在的 280 万[1]。同时，市场上智能手机的种类和品牌也越来越多，智能手机换代越来越频繁，智能手机的性能和质量也良莠不齐，这就带来了如下问题: 部分移动应用在开发时没有考虑智能手机本身的性能状况和传感器功能的完整性，其中存在大量的复杂计算，很多智能手机可能无法高效处理这些计算任务，使得应用运行效率低下，手机运行速度缓慢，严重的可能会威胁电池寿命。

本文设想了下面两个场景。

场景一，某用户正在用智能手机玩一款手机游戏，这个游戏中包含了人工智能（Artificial Intelligence，简称 AI）相关的计算， AI 计算任务往往需要很多的计算资源和很强的计算能力，而智能手机可能无法提供充足的计算资源进行运算，导致游戏运行缓慢，影响体验。这是由于计算任务的复杂导致的问题。

场景二，移动应用开发过程中，开发人员往往希望用不同配置和型号的移动设备对应用进行测试，一个有效的方法就是使用模拟器对不同的设备进行模拟，然而当应用涉及传感器数据的使用时，由于模拟器难以搭载传感器的功能，这样的测试往往无法实现。这是由于智能手机传感器功能缺失导致的问题。

为了解决场景一所示的问题，有一种比较成熟的方法是将移动应用内复杂的计算任务卸载，转移到远程服务器上运行，并将得到的结果返回至原应用。这种方法被称为 Computation Offloading，可以理解为计算任务的卸载与转移执行。

这种方法也存在着一些问题，比如使用远程的服务器提供计算资源需要手机与服务器保持稳定的连接状态，这一点在复杂的网络环境中难以做到。

也有研究人员给出不同的解决思路，即将计算任务转移到其他移动设备而不是远程服务器以避免 Offloading 需求出现在不稳定的网络环境中。

在场景二所示的传感器故障或功能缺失这个问题上，现有的工作多数研究智能手机的遥感或者传感器数据共享，很少有工作直接进行传感器功能的借用。

CoseDroid[2]是一个面向局域网并同时支持移

动应用计算任务转移执行和移动设备传感器功能借用的框架，其中传感器功能的借用称为 Sensing Offloading。在 CoseDroid 框架中，移动设备成为计算资源和传感器资源的提供者，同时 CoseDroid 给出了移动应用的插桩方案，解决了移动设备计算能力不足和传感器功能故障的问题。

CoseDroid 虽然是由移动设备提供计算资源和传感器资源，但是设备之间建立连接的过程和资源借用的调控仍然需要一个独立的服务器协调和维护，这使得在不同的局域网中需要维护不同的独立服务器，降低了框架的实用性。

一种直观的想法是，移动设备之间不通过独立的服务器，而是通过互相发现的方式建立连接，并且通过这种方式，移动设备间可以建立起一个互连网络，网络中每一个移动设备都可以作为资源的提供者，也可以作为资源的请求者；这种移动应用计算需求和传感器需求的 Offloading 也启发我们，应用对于屏幕资源的需求，即触控，亦可以进行转移，从而满足远程触控和多屏幕同时控制等使用需求，本文将这种 Offloading 称为 Touch Offloading。

结合上述思考，本文提出了 MultiOffload，一种支持多种资源借调的安卓应用 Offloading 平台；MultiOffload 面向局域网中的移动设备，搭载 MultiOffload 平台的移动设备可以自动地发现和连接其他设备，形成一个 P2P 网络，平台上的移动应用具有“借用”其他设备计算资源、传感器资源以及屏幕资源的能力。同时本文根据 MultiOffload 的架构设计重新实现和完善了 CoseDroid 架构中 Computation Offloading 和 Sensing Offloading 的插桩方案并基于程序插桩技术给出了触控转移（Touch Offloading）的实现方法。

在实验部分，本文实现了一个“Android 应用 Offloading 体验平台”以说明 MultiOffload 平台设计的可行性，选取了开源五子棋和涂鸦跳跃两个移动应用做插桩并展示 Computation、Sensing 和 Touch Offloading 的效果，同时选取了三个棋类游戏展示 Computation Offloading 对应用运行效率的提升。

综合而言，本文有如下两个创新点：

1.MultiOffload 是一个 P2P 平台，搭载平台的每个移动设备都可以借出自己的计算、传感器、屏幕资源或者借用其他设备的资源。

2.本文进一步拓展了 Offloading 的概念，将移动设备屏幕作为一种资源，提出屏幕资源的 Offloading，即 Touch Offloading，并结合已有的 Computation Offloading 和 Sensing Offloading 技术，给出了一个支持多种资源借调的安卓应用 Offloading 平台的设计方案。

本文的组织结构如下：第二节详细说明 MultiOffload 架构设计和平台中设备互连的实现方法；第三节介绍对 CoseDroid 架构 Computation Offloading、Sensing Offloading 技术的重新实现与完善，并提出 Touch Offloading 技术的实现方法；第四节介绍针对 MultiOffload 设计实现的一个示例，并选取应用展示 Offloading 技术的实现效果；第五节介绍本文的相关工作；第六节总结全文并对未来研究方向进行展望。

2 MultiOffload 平台设计

MultiOffload 是一个面向局域网的 Android 应用 Offloading 平台。文章之前提到，现有工作多数使用一个特定的服务器作为计算资源的提供方，这种方法存在一些问题，归纳如下：

问题一：服务器往往与移动设备处于不同的系统环境中，这会导致一些在 Android 环境中可以被运行的代码被转移到服务器环境中无法运行。

问题二：在局域网环境中，特定的服务器需要有固定的 IP 地址和配置。当局域网环境变化时（比如迁移到另一个局域网），之前网络中的服务器配置可能难以适配到现在的局域网，使得设备需要和服务器重新连接，服务器也需要重新配置。

问题三：移动设备的 Offloading 需求常常会发生变化，比如对于不同的移动应用，可能有不同的计算任务，需要不同的代码片段来执行，预先在服务器中准备好这些代码是极其困难的。

CoseDroid 框架解决了第一和第三个问题，对于问题二，由于 CoseDroid 使用了一个独立的名服务器（Name Server）以建立和维护设备之间的连接，当局域网本身发生变化时，需要重新维护和配置名服务器，为框架的实用性带来了问题。

基于对上述问题的思考，本文将 MultiOffload 设计为一个以设备发现为连接主导的 Offloading 平台。在这个平台上，每一个移动设备既可以成为建立互连网络的发起者，也可以成为建立互连网络的响应者。

图 1 展示了 MultiOffload 架构。下面将详细描述 MultiOffload 的架构设计和平台上各个设备之间建立连接的过程；架构中移动应用 Offloading 的插桩技术将在下一节讲述。

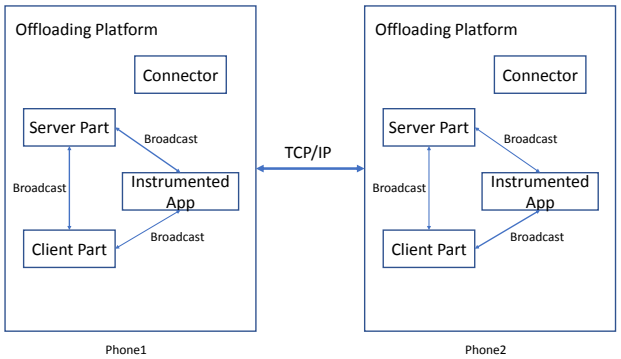

图 1 MultiOffload 架构
Fig.1 Architecture of MultiOffload

2.1 MultiOffload 平台架构

MultiOffload 平台由四部分构成：客户端部分（Client Part）、服务器部分（Server Part）、插桩后的移动应用（Instrumented App）和连接器（Connector）。

搭载 MultiOffload 平台的移动设备通过连接器互相发现并建立 TCP 连接，形成一个 P2P 网络。而 MultiOffload 内部采用客户端/服务器（Client-Server）架构，客户端部分承载了显示图形界面、开启应用等功能，服务器部分负责处理客户端的请求，并反馈给客户端和需要 Offloading 的移动应用。

在平台内部，客户端部分、服务器部分、移动应用之间通过广播（Broadcast）互相传递请求和响应；在互连的移动设备之间，客户端部分、服务器部分通过 TCP 连接发送请求和响应。这样每台移动设备都可以提供资源或请求资源。

这种架构设计解决了之前归纳的三个问题。

对于问题一，在本文的架构中，每个移动设备都可以作为服务器提供资源，设备之间系统环境基本相同，所以不会出现由于系统环境原因导致代码片段无法执行的情况。

对于问题二，移动设备以互相发现的方式建立连接，这就是说移动设备之间可以随时连接、随时断开，这非常符合移动设备网络环境易变动的特点，而且每台移动设备都可以作为服务器，服务器环境不需要静态配置。

对于问题三，本文和 CoseDroid 一样给平台客户端部分增加了文件发送的功能，请求计算资源的设备会提前准备好需要执行的代码片段，然后在请求服务时发送给服务设备，然后由服务设备动态加载执行该代码片段。

对这三个问题的解决说明本文设计的 MultiOffload 平台可以很好地处理设备互连和计算任务转移的相关问题。下面本文将说明 MultiOffload 如何让设备之间互相发现和连接。

2.2 设备发现与设备互连

局域网中搭载 MultiOffload 平台的移动设备之间的发现和互连是由连接器完成的，如图 2 所示，连接器由四部分组成：搜索管理器（Search Manager）、设备搜索器（Searcher）、连接建立器（Connect Creator）和搜索响应器（Responser）。

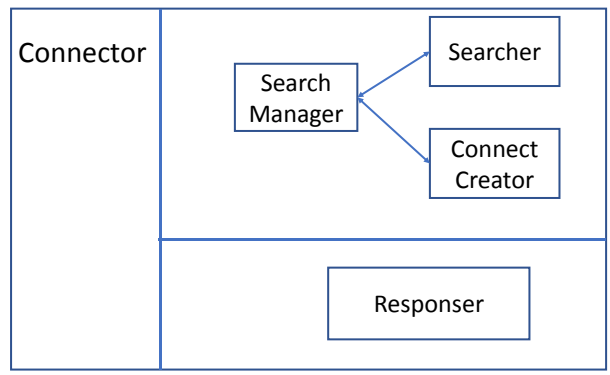

图 2 连接器组成
Fig.2 Connector Components

如图所示，这四部分赋予了移动设备两个可选角色，一个是作为主机的搜索者，另一个是等待主机搜索的响应者，下图（图 3 和图 4）完整地展示了设备之间建立连接的流程。

为了实现局域网中设备之间的互相发现，连接器的实现使用了 UDP 广播的方法。由于 UDP 本身是不可靠的连接协议，连接器在每次使用 UDP 广播进行通讯的时候都模拟了类似于 TCP 连接的“三次握手”协议。下图所示的连接流程可以分为三个阶段，其中前两个阶段都会经历模拟的“三次握手”协议。

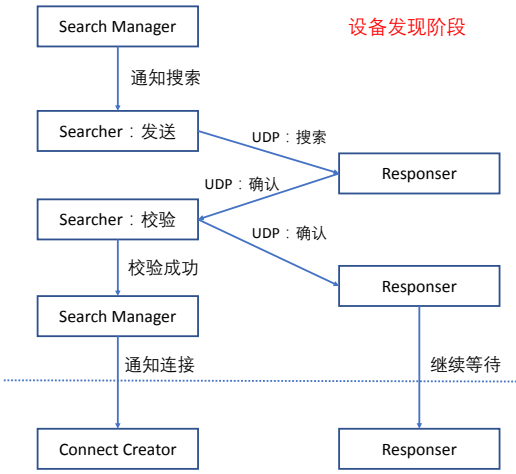

图 3 设备发现

Fig.3 Device Finding

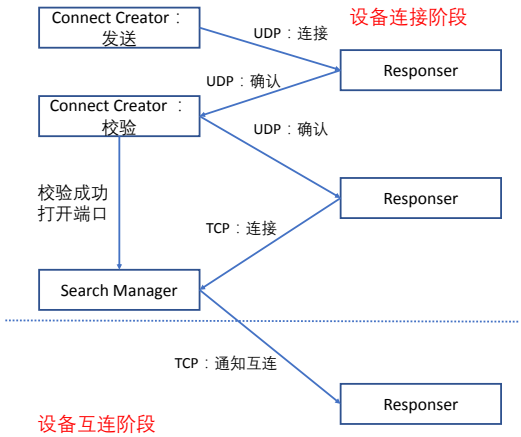

图 4 设备连接与设备互连

Fig.4 Device Connection and Device Interconnection

阶段一：设备发现

当一个设备选择作为主机进行搜索时，搜索管理器会通知设备搜索器进行搜索，即向局域网内发出 UDP 广播，这个广播附带自身编号和搜索器预先设定的广播类型（例如类型“搜索广播”）；此时，选择作为响应者的设备会开启搜索响应器，搜索响应器接收到主机发出的 UDP 广播并对广播内容进行校验，这是“第一次握手”。

响应设备校验成功后，搜索响应器会发出一个带有校验信息的 UDP 广播，主机在接收到这个广播后会再次进行校验，这就是“第二次握手”。

主机的设备搜索器校验成功后，会再次发出广播，并通知搜索管理器发现了想要连接的设备，并且获得该设备的 IP 地址等信息，使得搜索管理器开始第二阶段的连接部分，这是“第三次握手”。

阶段二：设备连接

在确认可以连接的移动设备的 IP 地址后，主机的搜索管理器会通知连接建立器进行第二轮的 UDP 广播。此时连接建立器会发出一个携带信息的 UDP 广播，信息内容即为响应者的 IP 地址。响应者在接到 UDP 广播后，会首先校验广播中要连接的 IP 地址是否为自身的 IP 地址，如果不是，就忽略广播报文；如果是，就发送回复的 UDP 报文，完成“第一次握手”。

主机接到回复广播后，校验其 IP 地址，完成“第二次握手”，随后发送回复广播，广播报文中会携带即将打开的新端口的端口号，并打开新的套接字（Socket）端口，等待响应者连接。

响应者接到回复的 UDP 广播，进行校验，完成“第三次握手”；并获得主机的 IP 地址和开放端口号，进而与主机建立 TCP 连接。由于 TCP 连接本身是可靠连接，所以不需要进行额外校验。这样，通过前两个阶段，主机就可以发现响应主机搜索的设备并建立起稳定连接。

阶段二描述的是两个设备之间的连接过程，第三阶段会建立多设备之间的连接网络。

阶段三：设备互连

主机在与响应设备建立连接后，会将与主机相连的所有设备的 IP 信息通过 TCP 连接发送给响应设备；同时主机会将响应设备的 IP 信息发送给每个与之相连的移动设备，让每个设备都打开新的 Socket 端口，等待响应设备进行连接。

响应设备接到主机发来的网络中所有设备的 IP 信息后，会逐一连接每个设备，与每个设备都建立 TCP 连接。这样就在原来的互连网络中新增了一台移动设备，同时在此互连过程中，响应者也会获得一个唯一标号或标识来确定其在网络中的身份。

注意到，在响应者与其他设备建立连接后，它们在网络中的地位都是一样的，主机和响应者只是不同设备在建立连接时所采用的不同角色。这种连接方式使每台设备都可以成为连接的发起者，也避免了连接网络中单点失效的问题。

在这三个阶段中，我们两次模拟“三次握手”协议，避免了使用 UDP 广播可能带来的网络不可靠问题。

3 Offloading 技术实现

移动应用一般不具有 Offloading 自身计算任务和传感器需求的能力，我们对移动应用进行插桩，使其具备这样的能力，并配合平台中的其他组件一起满足自身 Offloading 的需求。

插桩（Instrumentation）是分析和处理程序的常用技术，程序员通常可以在程序内部插入一些自己的代码，这些代码可以提供程序运行时的信息，比如程序运行时的堆栈信息、代码覆盖率信息等。同样编程人员也可以对移动应用进行插桩，在应用中插入某些代码，实现特定的程序行为和程序逻辑。

图 5 展示了一般应用的插桩流程。

本文针对字节码插桩使用的工具是 Soot[3]。Soot 是一个程序分析框架，它一开始是用来对 Java 程序做静态分析从而进行代码优化，现在也可以用来进行移动应用代码的插桩。Soot 可以接收应用的源代码，也可以接收应用的字节码，但是由于很多真实应用没有开源，源码不便获取，所以编程人员在插桩时一般使用的是 Soot 对字节码的处理能力。

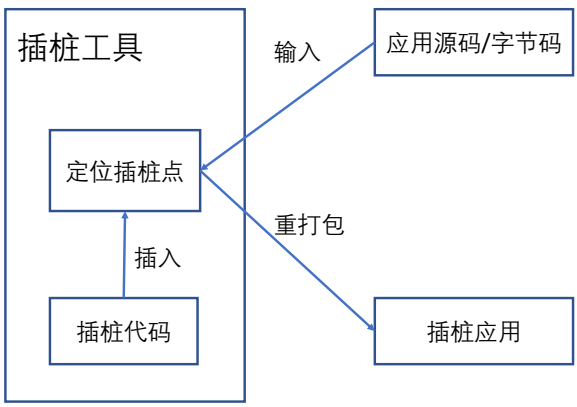

图 5 应用插桩流程
Fig.5 Application Instrumentation Process

Soot 会把 Java 字节码或者是 apk 文件转换成一种三地址中间代码，称为 Jimple 代码；同时 Soot 也封装了一些使用 Java 代码编写 Jimple 代码的接口，从而可以通过编写 Jimple 代码修改程序行为逻辑，屏蔽了直接对字节码插桩的复杂性。

本文的 Computation-、Sensing- 和 Touch-offloading 技术就是基于应用插桩实现的。

3.1 Computation Offloading 技术实现

Computation Offloading 技术希望可以将应用中部分涉及复杂计算的任务动态转移到高性能设备上执行，然后返回结果，使应用能够高效地运行。

这里本文结合 MultiOffload 架构对 CoseDroid 的插桩思想进行了重新实现。

CoseDroid 实现 Computation Offloading 技术的核心思想为：对于可转移的方法，记录其签名、所属类名、参数，序列化调用对象；将对象传输至服务设备通过 Java 反射机制动态加载字节码片段，执行该方法，并将执行结束后的结果和调用对象一同返回；最后将返回对象的每一个域拷贝给原对象。

基于上述目标，本文设计了如图 6 所示的插桩流程。

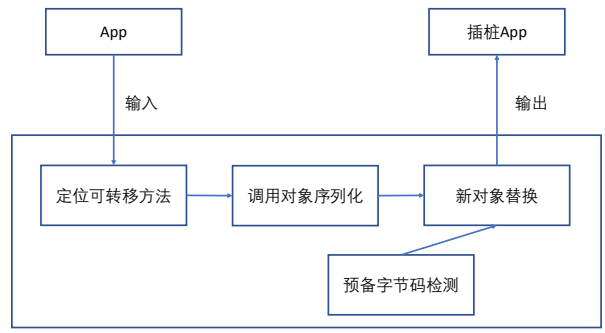

图 6 Computation Offloading 插桩流程
Fig.6 Computation Offloading Instrumentation Process

并不是所有的方法都是可转移的，可转移的方法要求方法执行前后只对调用对象本身的域（Field）造成影响，不会对系统状态和其他程序状态造成影响。对于类C中的方法m（带有引用对象的参数p）是否可转移，本文结合 MultiOffload 架构将其总结为表 1 所示的四个条件（判定结果为“是”则不满足条件）。

流程中插入了预备字节码检测环节，其意义在于，当计算资源的提供设备没有应该执行的字节码片段时，该环节会提醒 MultiOffload 平台及时发送所需字节码片段给计算设备；如果请求设备也没有预备应该执行的字节码片段，该环节则会终止 Offloading 请求，使该方法本地执行，以免产生传输错误，保证了 Computation Offloading 请求过程的鲁棒性。

表 1 方法可转移条件

Table.1 Conditions of Offloadable Methods

条件	描述	判定方法
可序列化	类C及其父类已实现或可实现 Serializable 接口	判断类C及其父类是否是 Android 框架类的子类
I/O-系统调用	m的方法体中不存在 I/O 操作和 Android 系统调用	对于m方法体的每个语句，判断是否包含 I/O 或 Android 系统实例或调用
参数引用修改	方法m没有修改参数p中引用对象的域	对于m方法体的每个赋值语句，判断其左值是否是p的域或p中的某个元素；对于包含方法调用的语句则递归判断
静态域使用-修改	方法m没有使用和修改类的静态域	对于m方法体的每个语句，判断其是否包含类C及其他类的静态不；对于包含方法调用的语句则递归判断

3.2 Sensing Offloading 技术实现

Sensing Offloading 技术希望某些因为移动设备传感器故障或者功能缺失而不能运行的应用，可以借用其他设备的传感器功能，通过其他设备的传感器获得数据，以维持应用的正常运行。

为了说明 Sensing Offloading 的插桩方法，这里简介 Android 应用是如何使用传感器的。Android 应用的开发工具 Android SDK 实现了一个 Android 传感器框架（Android Sensor Framework[4]，简称 ASF），里面包含着使用设备传感器的接口。

Android 应用可以通过 ASF 提供的 getSystemService 方法获得 SensorManager 类的实例对象，再通过 getDefaultSensor 获取希望使用的传感器实例（如重力传感器的编号是 TYPE_GRAVITY），在获得传感器实例后，Android 应用可以通过 registerListener 和 unregisterListener 两个方法注册和注销 SensorEventListerner 接口实现对传感器数据的获取、精度变化的感知等。

CoseDroid 提出通过监控 registerListener 和 unregisterListener 两个方法获得 Android 应用传感器的注册/注销信息，并使用改写的 SensorEventListerner 实例替换原应用注册的传感器事件处理器，从而实现传感器的借用。

本文根据上述思想设计了如图 7 所示的插桩流程。

经过这两步的插桩，插桩应用在运行时会注册预先写好的一个传感器数据处理器，而这个处理器会从提供传感器服务的设备上获取数据，然后返回给应用，使得插桩应用检测到运行设备具有了所需传感器的功能。

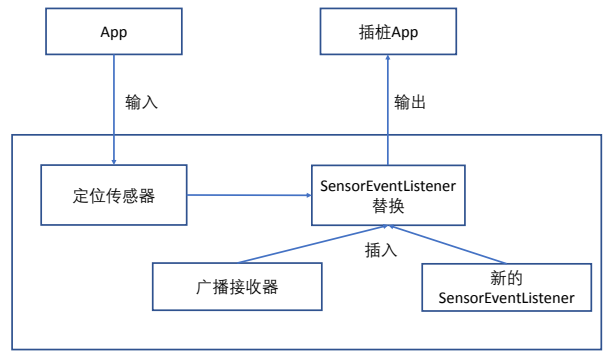

图 7 Sensing Offloading 插桩流程

Fig.7 Sensing Offloading Instrumentation Process

3.3 Touch Offloading 技术实现

移动应用计算需求和传感器需求转移的提出使我们想到移动应用对于屏幕资源的需求，即触摸控制（Touch）也可以被转移，其意义在于实现远程触控，例如以一台设备作为控制板控制多台设备进行同步演示等。本文把这种 Offloading 称为 Touch Offloading。

对于 Touch Offloading 技术的实现，本文设计了图 8 所示的插桩流程。

在移动应用中，onTouch 方法是其处理触控指示的方法。插桩工具会定位到这个方法，并将其替换为预先编写好的 onTouch 方法，同时给应用插入的广播接收器会接收触控设备发出的触控信息，即触屏的具体坐标，从而使用 MoveEvent 类（Android 架构中触摸事件类）触发对 Offloading 设备的触摸操作，完成触控设备对 Offloading 设备的远程触控。

值得关注的是，本文对移动应用的插桩是一种

轻量级的插桩，不会影响在没有 Offloading 平台下移动应用的运行。

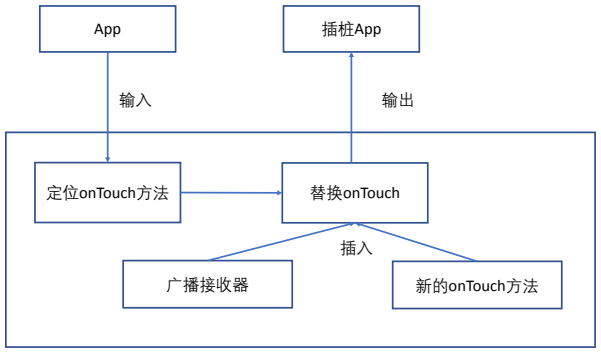

图 8 Touch Offloading 插桩流程
Fig.8 Touch Offloading Instrumentation Process

所以，在三种 Offloading 技术的插桩中本文都加入了关于执行模式的判断，如果应用不是由 MultiOffload 平台打开运行的，那么：

被 Offloading 的方法不会做任何处理，在原设备上执行；应用原来的传感器事件处理器不会被替换，继续获取本机上的传感器数据；管理触控的 onTouch 方法不会被修改，依然可以触摸设备屏幕对应用进行控制。

当用户正常打开插桩后的 Android 应用时，应用会如没有被修改过一般正常运行。

4 实验评估

4.1 实验设计与实现

为了说明 MultiOffload 平台设计的可行性，我们开发了一个“Android 应用 Offloading 体验平台”（简称为 Offloading 体验平台）。图 9 展示了 Offloading 体验平台的主界面，可以看到，主界面除了启动应用、建立连接、打开说明和改变计算任务转移目标外，还可以显示设备是否已经互连形成 Offloading 网络，如果已互连，那么顶端会显示当前设备的编号。

在 Offloading 体验平台上，本文选取了两个移动应用来展示 Offloading 技术的实现效果。

对于 Computation Offloading 的展示，本文选取开源五子棋游戏作为展示应用。在这个五子棋应用中存在一个基于人工智能算法的方法，这个方法用于计算电脑下一步的最佳落子点，这个方法的执行效率影响着整个应用的运行效果，非常适合展示计算任务的转移执行对程序运行效率的影响。

对于 Sensing Offloading 和 Touch Offloading 的展示，本文选择涂鸦跳跃（Doodle Jump）作为展示应用。涂鸦跳跃是一款利用加速度传感器控制人物保持跳跃并躲避障碍的游戏，这款游戏只使用了加速度传感器，所以用来展示传感器资源的借用非常合适，它本身还有比较简洁的操作界面，也适合于展示触控转移。

除了验证 MultiOffload 平台设计和 Computation、Sensing、Touch Offloading 插桩技术的可行性，本文还希望探究 Computation Offloading 后 Android 应用的运行效率提高的程度，即应用运行时间的减少程度。

本文在开源五子棋应用之外又挑选了 Github 上两个棋类游戏 GoBang（五子棋）和 Othello（黑白棋），通过比较这三个游戏 Offloading 前后连续十步棋的平均时间，观察应用运行时间是否减少和减少的程度。

图 9 Offloading 体验平台主界面
Fig.9 Home Screen of Offloading Experience Platform

结合上述实验目的，本文提出三个研究问题。

问题一：将 Android 应用的复杂计算转移给更高性能的设备执行是否可行，相同步骤内应用运行时间是否明显减少。

问题二：本文 Sensing Offloading 的插桩方法是

否可以使 Android 应用正确接收其他设备的传感器数据进而正常运行。

问题三：本文 Touch Offloading 插桩技术是否能够实现触控转移，即在控制者屏幕上进行点击是否能在显示者屏幕上相同位置产生点击事件（转移触控的设备称为控制者，接受控制的设备称为显示者）。

在展示实验结果之前，我们先说明本文的实验环境，本文使用两台三星 Galaxy S5、一台华为 P6、一台华为 Mate7 在 1167Mbps 的 WIFI 局域网下进行实验。

4.2 实验结果

4.2.1 Computation Offloading 效果展示

对于问题一，本文选用华为 P6 和华为 Mate7 两台设备进行实验，根据配置说明，Mate7 性能明显高于 P6。图 10 和图 11 分别展示了在 P6 设备上运行开源五子棋在 Offloading 前后一步棋的所需时间。为了更直观地展示实验效果，我们在插桩时重复多次运行 Offloading 的方法，在开源五子棋应用的展示中，Offloading 的方法被执行了 500 次，在后面的实验中我们也会对其他应用 Offloading 的方法重复执行合适的倍数。

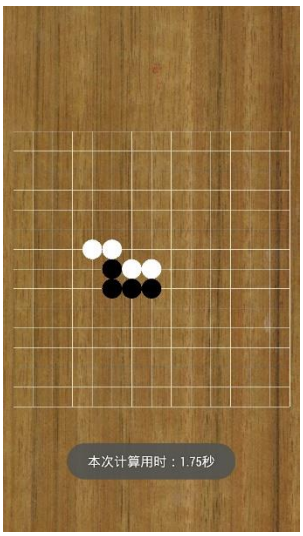

图 10 Offloading 前 P6 运行五子棋的一步耗时
Fig.10 One Step Time Consumption on P6 Before Offloading

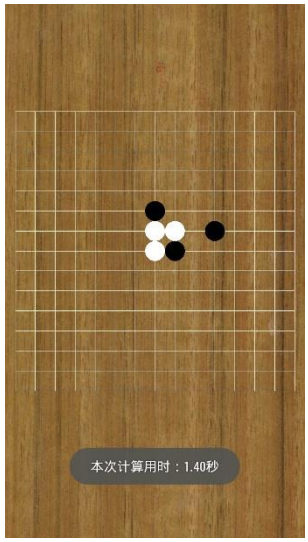

图 11 Offloading 后 P6 运行五子棋的一步耗时
Fig.11 One Step Time Consumption on P6 After Offloading

实验中五子棋游戏运行正常，未出现游戏崩溃的情况。

为了探究应用运行时间的减少，本文对 GoBang 和 Othello 做了类似的插桩，比较 Offloading 前后连续十步棋的平均时间，得到表 2.

表 2 三个棋类游戏 Offloading 前后耗时比较
Table.2 Time Consumption of Three Chess Games Before and After Offloading

实验对象	P6 本地执行（s）	计算转移至 Mate7（s）	减少比例	备注
开源五子棋	1.79	1.35	24.6%	重复执行 500 次
GoBang	2.14	1.88	12.1%	重复执行 100 次
Othello	2.51	2.03	19.1%	重复执行 100 次

从表 2 可以看出，Computation Offloading 对 Android 应用运行时间的减少具有可观的效果。

4.2.2 Sensing Offloading 效果展示

对于问题二，本文使用两台三星设备进行实验，其中提供传感器数据的设备称为控制者，接收传感

器数据的设备称为显示者。我们左右摇晃控制者，显示者中由加速度传感器控制的人物跟随控制者进行基本无延迟的左右跳跃；而左右摇晃显示者未对游戏造成任何影响。

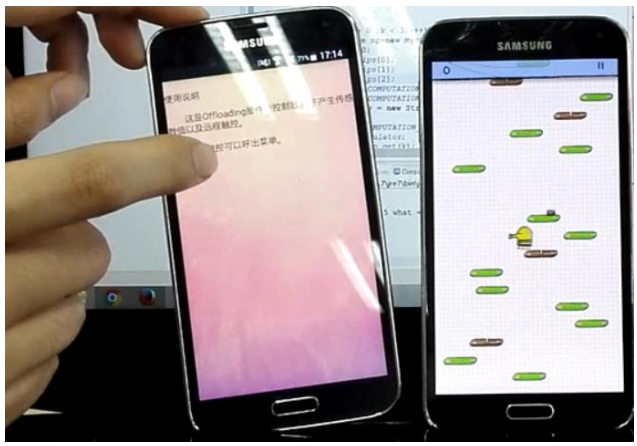

图 12 Sensing Offloading 效果展示
Fig.12 Sensing Offloading Effect

图 12 展示了控制者（左）向左摇，游戏中人物（右）跟随着向左跳跃的情景。说明文中 Sensing-offloading 的插桩方法能够使 Android 应用成功借用其他设备的传感器功能。

4.2.3 Touch Offloading 效果展示

对于问题三，本文同样使用两个三星设备进行实验。为了能够确认控制者上的点击信息输到了显示者，我们在插入广播接收器时注册了额外的广播信号，使得显示者在接收到控制者的点击信息时，以点击点为中心绘制圆环。这样既可以看到控制者上的点击是否传递给了显示者，又可以判断显示设备接收到的数据是否正确。

实验时，我们在控制者的控制板上进行随意点击，同时观察显示者屏幕上的圆环位置，结果圆环中心与显示者屏幕的相对位置，与点击点与触控设备屏幕的相对位置基本一致；我们在控制板上点击显示者涂鸦跳跃游戏界面上的按钮的相对位置，显示者的游戏界面做出和直接点击按钮一样的跳转。图 13 展示了点击触控设备控制板后，显示设备在同样的相对位置出现圆环的情景。

这说明本文给出的 Touch Offloading 插桩方法可以有效实现触控转移。

综合上述三个实验结果，我们可以得出结论，本文设计的 MultiOffload 平台架构具有可行性，本文重新实现并完善的 Computation Offloading 和 Sensing Offloading 插桩技术、提出的 Touch Offloading 插桩技术可以使 Android 应用具有借用其他设备资源的能力，并且 Computation Offloading 对 Android 应用的运行时间有可观的减少。

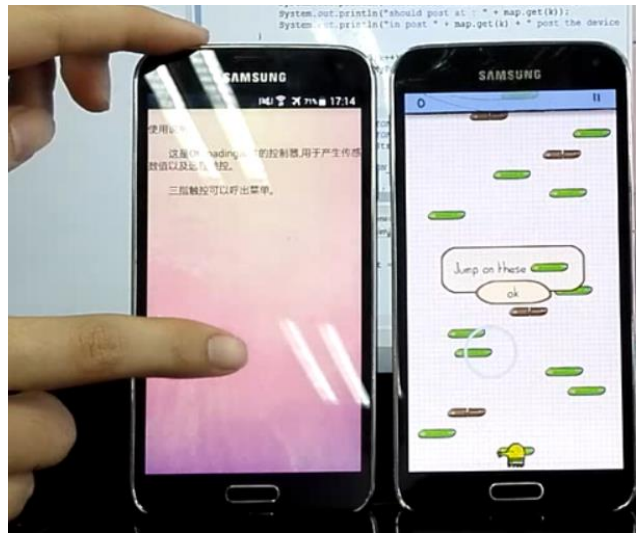

图 13 Touch Offloading 效果展示
Fig.13 Touch Offloading Effect

5 相关工作

本文提出的 MultiOffload 是一个同时支持 Computation Offloading、Sensing Offloading 和 Touch Offloading 的 Android 应用 Offloading 平台，而对于 Offloading 技术也已经有许多相关研究。

起初关于 Offloading 的研究主要针对于个人电脑（Personal Computer，简称 PC）应用，由于服务器的性能明显优于 PC 的性能，并且部分 PC 程序逻辑复杂，计算任务重，因此将 PC 应用繁杂的计算转移到服务器上运行可以明显地提升性能。Javaparty[5]和 J-Orchestra[6]关注 PC 上 Java 应用的远程执行，他们使用用户标注和程序分析的方法，基于 Java 的远程方法调用（Remote Method Invocation，简称 RMI）技术，将需要计算的方法所属对象传输至服务器上执行，但因为 Android 平台不支持 RMI 技术，所以这两份工作难以直接移植到 Android 平台。

也有一些对非 Android 平台的应用 Offloading

技术的研究工作。Maui[7]就是面向 Windows Phone 平台的应用和设备的 Offloading 工作，工作语言和处理对象是 C#语言和.NET 程序，其工作核心是远程过程调用（Remote Procedure Call，简称 RPC），在运行时决定哪些方法应该转移执行，然后做程序分割；Maui 提供了一个编程环境并允许用户标注哪些方法可以被转移执行，并在运行时判断这个方法是否真的可以 Offloading。L. Wang[8]的工作面向支持 Java 的移动设备，这份工作的重点在于面向对象程序中的程序分割，使用静态分析的方法首先建立一个带权值的对象关系图，然后根据这个对象关系图二次建模将其中的对象划分成本地执行的部分和服务器执行的部分，并对划分成服务器部分的对象进行转移执行。

在这些针对非 Android 平台的工作出现的年代，Android 平台还没有成熟。现在流行的移动平台是 Android 平台和 iOS 平台，由于 iOS 系统闭源等特点，iOS 应用难以分析和处理；相比之下，Android 是一个开源系统，存在许多 Android 应用的分析工具，比如 Soot，所以现在大部分工作都集中在 Android 平台上，尤其是 Android 应用的 Computation Offloading。

Cuckoo[9]提供了一个 Android 应用 Computation Offloading 的编程模型，它要求开发者在这个编程模型下开发应用。在 Cuckoo 的编程模型中，开发人员需要在本地的“服务（Service）”部分定义需要 Offloading 的方法接口，这个 Service 是支持 RPC 的，同时 Cuckoo 框架会自动在远程服务器生成本地 Service 中接口的拷贝；在应用开发时，开发人员在本地 Service 中实现之前定义的方法接口，覆盖远程 Service 中的接口；进而在应用运行调用这些方法时，会主动请求服务器执行这些方法，完成计算的 Offloading。与 Cuckoo 相似的工作还有 E. Chen[10]，但是它不需要开发者在特定的编程模型下开发，而是处理以特定开发模式开发的 Android 应用，对于每个用户，E. Chen[10]都在云端服务器上部署一个虚拟移动设备，并将计算任务转移到虚拟设备上运行。上述两份工作对 Offloading 技术的实现主要依靠 Android 应用的开发模式，而 DPartner[11]是通过字节码重写的方式重构一个 Android 应用，支持不同粒度的计算转移。

在 Computation Offloading 研究中，也有工作不是进行方法上的 Offloading，而是从应用内部线程、服务器资源、能耗等多方面考察 Offloading 的可行性。CloneCloud[12]和 COMET[13]关注于多线程 Android 应用中线程的 Offloading。它们不是方法驱动的 Offloading 策略，而是线程驱动的策略；COMET 没有对应用字节码做修改，而是修改 Dalvik 虚拟机，其所有操作都建立在 Java 内存模型上，并使用分布式共享内存模型，提高了 Offloading 效率。COSMOS[14]关注云服务器资源的调度和网络连接情况的权衡，而 A. Mukherjee[15]则先对应用本身 Offloading 的能耗做评估，以应用 Offloading 后的能耗是否减小为基准判断应用是否值得 Offloading。

在 Sensing Offloading 的研究上，D. Yang 的[16]可以理解为一种 Crowd Sensing，它是让中央节点分发若干传感器相关的任务给不同的移动设备，然后移动设备可以接受任务并完成，也就实现了传感器的合作。D. Yang [16]的关注点在于设备之间的利益问题，而本文的工作侧重于一对一的传感器服务，重点在于如何使得 Android 应用获得使用其他设备传感器的能力。Metis[17]考虑的是社交行为中传感器的 Offloading，他们认为一直使用本机的传感器记录人的社交行为有很大的开销，于是他们考虑在环境中部署相应的传感器，进而可以使用这些环境中的传感器替代本机的传感器，达到节省开销的目的。Metis 通过实验归纳出一些适宜在环境中部署的传感器，并根据归纳结果提出了一个 Sensing Offloading 框架，提供了一种编程模式，用于与社交行为有关的 Android 应用的开发。与本文工作不同的是，本文通过对 Android 应用进行轻量级的插桩，就可以使应用具有 Offloading 传感器需求的能力，但是如果希望得到一个可以和环境传感器交互的 Android 应用，往往需要开发人员依照 Metis 框架重新开发，很难通过修改应用得到。

在多种 Offloading 同时支持的研究问题上，CoseDroid[2]是一个同时支持 Computation 和 Sensing 的 Android 应用 Offloading 框架，关注计算任务和传感器 Offloading 后的能耗降低问题；与本文的 MultiOffload 不同的是，CoseDroid 在移动设备建立互连时仍然使用了一个名服务器为媒介，设备之间依靠独立的服务器建立连接，同时本文对 CoseDroid 的 Offloading 技术做了拓展，提出了 Touch Offloading 技术，即触控转移。

6 总结与展望

本文给出了一个 Android 应用 Offloading 平台的设计与实现方案，MultiOffload；MultiOffload 同时支持 Android 应用进行 Computation Offloading、Sensing Offloading 和 Touch Offloading，搭载 MultiOffload 的移动设备可以互相发现并建立连接，形成一个 P2P 网络。

本文重新实现和完善了已有工作的 Computation Offloading 和 Sensing Offloading 插桩技术，并提出 Touch Offloading 的轻量级程序插桩实现方案；使得插装后的 Android 应用具有触控转移的能力和借用 P2P 网络中移动设备计算资源、传感器资源的能力。

本文的工作还有继续完善的空间，触控是 Android 应用进行响应的最基本的输入。除此以外，有些 Android 应用还会响应手势、摄像机和指纹等输入，实现指纹输入的 Offloading 有利于实现远程解锁等功能。未来，我们会逐步实现手势、摄像机、指纹的 Offloading，扩展 Offloading 的概念，实现 Android 应用控制的转移。

References:

[1] “Number of applications in Google Play Store”, https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[2] X. Wu, C. Xu, Z. Lu, Y. Jiang, C. Cao, X. Ma, and J. Lu. CoseDroid: Effective computation- and sensing-offloading for Android apps. In Proceedings of the 39th Computer Society International Conference on Computers, Software and Applications (COMPSAC), pp. 632-637, 2015.

[3] “Soot”, http://sable.github.io/soot/.

[4] “Sensor Overview”, https://developer.android.com/guide/topics/sensors/sensors_overview.html.

[5] M. Philippsen and M. Zenger. Javaparty - Transparent remote objects in java . In Concurrency: Practice & Experience, Vol.9, No.11, Wiley, pp.1225–1242, 1997.

[6] E. Tilevich and Y. Smaragdakis. J-Orchestra: Automatic Java application partitioning. In Proceedings of European Conference on Object Oriented Programming (ECOOP’02), pp.1-3, 2002.

[7] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making smartphones last longer with code offload. In Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys), pp.49-62, 2010.

[8] L. Wang and M. Franz. Automatic partitioning of object-oriented programs for resource-constrained mobile devices with multiple distribution objectives. In Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS), pp. 369-376, 2008.

[9] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: A computation offloading framework for smartphones. In Proceedings of the 2nd International Conference on Mobile Computing, Applications, and Services (MobiCASE), pp.59-79, 2012.

[10] E. Chen, S. Ogata, and K. Horikawa. Offloading Android applications to the cloud without customizing android. In Proceedings of the 10th Internation Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp.788–793, 2012.

[11] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang. Refactoring Android java code for on-demand computation offloading. In Proceedings of the ACM Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA), vol. 47, no. 10, pp.233-248, 2012.

[12] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: Elastic execution between mobile device and cloud. In Proceedings of the 6th European Conference on Computer Systems (EuroSys), pp.301–314, 2011.

[13] M. S. Gordon, D. A. Jamshidi, S. A. Mahlke, Z. M. Mao, and X. Chen. Comet: Code offload by migrating execution transparently. In Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI), pp.93–106, 2012.

[14] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura. COSMOS:computation

offloading as a service for mobile devices. In Proceedings of the 15th ACM international symposium on Mobile AdHoc Metworking and Computing(MobiHoc), pp. 287–296, 2014.

[15] A. Mukherjee, and D. De. Low power offloading strategy for femto-cloud mobile network. In Engineering Science & Technology An International Journal, pp.260-270, 2016.

[16] D. Yang, G. Xue, X. Fang, and J. Tang. Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing. In Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (MobiCom), pp.173–184, 2012.

[17] K. K. Rachuri, C. Efstratiou, I. Leontiadis, C. Mascolo, and P. J. Rentfrow. Metis: Exploring mobile phone sensing offloading for efficiently supporting social sensing applications. In Proceedings of the 11th IEEE International Conference on Pervasive Computing and Communications (PERCOM), pp. 85–93, 2013.