软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2017,28(12):3367−3384 [doi: 10.13328/j.cnki.jos.005325]

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

# 一种优化安卓应用 3G/4G 网络请求能耗的方法*

蔡华谦[1,2], 张 颖[2,3], 黄 罡[1,2], 梅 宏[1,2]

[1](北京大学 信息科学技术学院 软件研究所,北京 100871)

[2](高可信软件技术教育部重点实验室(北京大学),北京 100871)

[3](北京大学 软件工程国家工程研究中心,北京 100871)

通信作者: 张颖, E-mail: zhang.ying@pku.edu.cn

**摘 要**: 智能手机后台应用的网络请求极大地影响着待机时间.已有的工作提出了节省手机能耗的应用网络请求调度算法,然而,如何将算法自动地应用到既有手机系统,仍面临着巨大挑战:(1) 在没有应用源代码的情况下,实现单个应用内的网络请求合并;(2) 在不对操作系统进行任何修改的情况下,按需合并多个应用中的网络请求.以安卓应用为目标,给出了一种通过自动程序转换来支持现有移动应用中网络请求延迟调度的方法及其框架实现——DelayDroid,用以提升手机整体待机时间.通过字节码分析和程序自动转换技术解决以上挑战.与已有工作相比,DelayDroid 有两大特色:一是程序转换自动执行;二是转换后的应用可支持多应用的后台网络请求调度,该调度机制可以降低安卓应用的待机耗电.此外,DelayDroid 被设计为可对只有 dex 字节码的安卓应用进行转换,更具实用性.

**关键词**: 网络请求合并;安卓应用优化;程序转换;能耗

**中图法分类号**: TP316

中文引用格式: 蔡华谦,张颖,黄罡,梅宏.一种优化安卓应用 3G/4G 网络请求能耗的方法.软件学报,2017,28(12):3367−3384. http://www.jos.org.cn/1000-9825/5325.htm

英文引用格式: Cai HQ, Zhang Y, Huang G, Mei H. Approach to scheduling network requests in Android apps. Ruan Jian Xue Bao/Journal of Software, 2017,28(12):3367−3384 (in Chinese). http://www.jos.org.cn/1000-9825/5325.htm

## Approach to Scheduling Network Requests in Android Apps

CAI Hua-Qian[1,2], ZHANG Ying[2,3], HUANG Gang[1,2], MEI Hong[1,2]

[1](Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Key Laboratory of High Confidence Software Technology (Peking University), Ministry of Education, Beijing 100871, China)

[3](National Engineering Research Center for Software Engineering, Peking University), Beijing 100871, China)

**Abstract**: Mobile devices with 3G/4G networking often waste energy in the so-called “tail time” during which the radio is kept on even though no communication is occurring. Prior work has developed policies to reduce this energy waste by batching network requests. However, it is challenging to apply such policies to existing apps in practice due to lack of mechanisms. This paper proposes an automatic program transformation approach for scheduling network requests in Android apps. The core of the approach is bytecode transformation for existing Android apps. Addressing the technical challenges in automatic transformation, the paper implements a transformation system named DelayDroid. Comparing to previous work, DelayDroid has two major characteristics. First, transformation is carried out automatically. Second, DelayDroid is designed to be a practicable tool, as it can transform Android apps with only dex bytecode.

**Key words**: network request batching; Android app optimization; program transformation; energy

* 基金项目: 国家高技术研究发展计划(863)(2015AA01A202); 国家自然科学基金(61421091)

Foundation item: High-Tech Research and Development Program of China (863) (2015AA01A202); National Natural Science Foundation of China (61421091)

收稿时间: 2016-06-26; 修改时间: 2017-01-11; 采用时间: 2017-07-06

随着智能手机的发展,移动应用越来越依赖云端提供的硬件、软件资源,以提供更好的服务[1].然而,云-端通信需要消耗大量的电能.联网应用(如天气、邮件、新闻等)呈现了网构软件[2]典型的构件化特点,利用网络实现了终端与云端各构件的通信.特别在 3G/4G 环境下,联网应用在后台长时间间隔地利用网络获取相应的推送消息.这种长时间间隔式的消息推送,给电池容量有限的智能手机的续航带来了巨大压力.针对移动联网应用的网络能耗问题,已有的工作提出了一系列的网络请求延迟发送、提前预取的合并算法[3−5].然而,面对现有大量的既有移动应用,它们处于软件生产的后期阶段[6],如何将这些算法部署其中,仍面临巨大挑战.

从操作系统架构的角度出发,部署网络请求合并算法的方法可以分为 3 类:应用层部署、框架层部署和系统层部署.

1) 应用层部署即由应用开发人员通过修改应用源代码的方式,将网络请求的合并调度算法实现到应用中.对于单个应用而言,这种实现方式可以最大限度地利用应用的上下文信息以节省网络能耗.然而,这种方式仅能对单个应用内部的网络请求进行合并,而智能手机的 3G/4G 网络模块是系统中的所有应用共享的.这种部署方式缺乏全局的统筹调度能力,虽然可以实现单一应用最优,却不能实现全局最优.此外,这种部署方式需要开发人员较大地改动应用源代码.

2) 框架层部署即由系统开发人员提供新的框架层应用编程接口(API),应用开发人员利用这些框架来发送网络请求,由框架负责合并来自不同应用的多个网络请求,以节省网络能耗.例如,谷歌的安卓开发框架在其 5.0 版本之后,提供了一组可以合并 3G/4G 网络请求的应用编程接口(PeriodicTask)[7].利用这组编程接口,应用开发人员可以实现跨应用的网络请求合并.然而,该部署方式不仅需开发人员修改应用的源代码,还需要特定操作系统的支持.根据 2016 年 1 月谷歌发布的安卓版本分布数据,安卓 4.4 的市场份额仍高达 32.5%;安卓 4.4 及之前版本 20%以上[8].因此,该方式难以应用到占比 52.5 以上的旧版本操作系统当中.

3) 系统层部署即修改网络相关的系统调用的具体实现,以实现网络请求的合并.由于所有的网络请求最终都会进行系统调用,因而修改系统调用能够实现全局网络合并的效果,也不需要应用开发人员修改应用.然而,由于缺乏应用的上下文信息,系统难以判断哪些网络请求需要合并,同时,也难以针对特定应用进行优化.

由此可见,现有工作难以同时满足:(1) 合并多个应用的网络请求以节省手机待机电能;(2) 降低开发人员工作量;(3) 提升适用范围,兼容旧版本操作系统.本文以广泛存在的安卓应用为对象,研究如何基于自动程序转换实现应用网络请求合并.本文提出了一种支持网络请求合并的机制,并着重介绍了将给定应用转换为支持网络请求合并的应用过程中所面临的主要技术挑战:(1) 在没有应用源代码的情况下,实现单个应用内的网络请求合并;(2) 在不对操作系统进行任何修改的情况下,按需合并多个应用中的网络请求.本文实现了名为 DelayDroid 的自动程序转换工具,将给定的安卓应用转换为可自动合并网络请求的安卓应用.DelayDroid 在 Dex 字节码[9]层次上实现转换,这使得它不仅可用于新开发的应用,而且可用于现有应用.DelayDroid 的输入为一个给定的安卓应用的安装包,经过字节码分析、重构和依赖库注入等步骤,输出一个优化后的安卓应用的安装包.当多个经过 DelayDroid 自动转换后的应用同时运行的时候,它们的网络请求会根据 DelayDroid 内置的合并算法进行合并.本文以安卓应用中的天气、邮件和新闻应用为例对 DelayDroid 进行了实验,并证明了合并网络请求后应用的能耗节省.本文的主要贡献在于:

1) 提出了一种基于自动重构实现应用网络请求的按需合并机制.利用该机制,开发人员可以方便地将自定义的网络请求合并算法实现到现有应用中;
2) 给出了该机制的一种具体设计,并实现了 DelayDroid 应用转换系统;同时,在 Google Play 应用商店和豌豆荚应用商店中选取了国内外的 12 个应用,验证了 DelayDroid 应用转换系统的实用性.提出了一种缺省网络请求合并算法,并利用 DelayDroid 将该合并算法实现到了一组现有的安卓应用中.实验结果表明:在待机状态下,利用该合并算法可以节省高达 32%的网络能耗.

本文第 1 节给出 3G/4G 网络耗电的特点以及一个安卓应用的网络请求合并的例子.第 2 节介绍如何自动程序转换实现安卓应用的网络请求合并.第 3 节介绍 DelayDroid 的具体实现,即,如何通过字节码级的自动程序转换而将给定的安卓应用转换为部署了网络请求合并算法的应用.同时,第 3 节还给出一个调度算法实例.第 4 节

通过一组实验,利用 DelayDroid 将第 3 节中的算法部署到一些真实应用中,证明 DelayDroid 的有效性.第 5 节介绍相关工作.第 6 节总结全文并展望未来的工作.

## 1 背景与网络请求合并实例

### 1.1 3G/4G网络耗电特点

3G 和 4G 是当前主流使用的移动蜂窝网络.3GPP(third generation partnership project)[10]和 4GPP[11]是当前主采用的 3G/4G 标准.相比于 WiFi 的耗电,蜂窝网络的耗电特点更为复杂:一方面,因为蜂窝网络移动性较强,对于一个移动设备,随着物理位置的移动,可能快速切换到不同蜂窝网络基站,因此,对于蜂窝网络基站,不可能将一个信道一直分配给一台移动设备;另一方面,由于移动设备其续航有限,而长时间连接至蜂窝网络基站会提高其功耗,影响续航,因此,蜂窝网络标准中进一步对无线资源控制(radio resource control,简称 RRC)模块的状态进行了规定.

以移动设备中的 3G 网络模块为例,一共包含 3 个状态,如图 1 所示.

1) IDLE:即空闲状态,在此状态下,3G 模块的耗电最低,同时也不能发送、接收任何数据.在这个状态下,如果要发送或接收数据,则会转移至 CELL_DCH 状态;
2) CELL_DCH:在这个状态下,3G 模块的带宽达到最大,此时能以最大的速率进行数据传输;同时,它的功耗也是最大的.如果持续一段时间仍然没有数据传输的话,它会转移至 CELL_FACH 状态.根据不同的运营商,持续运行于 CELL_DCH 状态的时间通常是 5s~10s;
3) CELL_FACH:在这个状态下,3G 模块的耗电比 CELL_DCH 要节省 50%;同时,在这个状态下,其网络传输速率也较低.如果在这个状态发送或接收的数据大于某个阈值,则会重新转移至 CELL_DCH 状态;而如果在 CELL_FACH 状态持续一段时间没有发送或接收数据,则会转移至 IDLE 状态.一般来说,这个时间通常是 10s~15s.

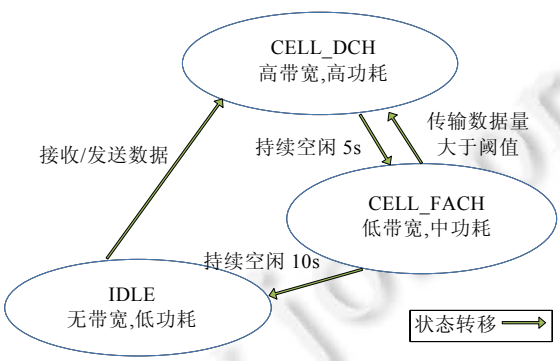



Fig.1 Radio resource control state machine

图 1 无线资源控制状态机

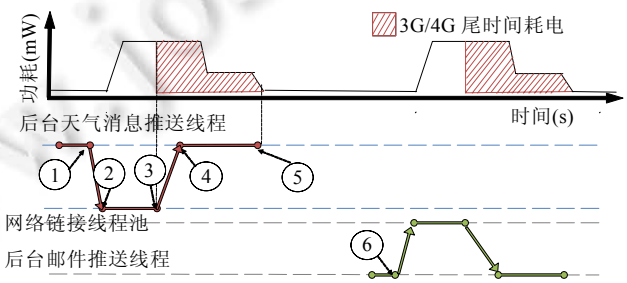
在实际使用中,在 IDLE 和 CELL_DCH 状态的切换需要 3s~4s 的时间,因此,该标准引入 CELL_FACH 状态是对网络延迟与网络耗电的一种折衷.然而,这个机制使得从 CELL_DCH 状态转移至 IDLE 状态所需的时间更长,我们称这段由 CELL_DCH 状态转移至 IDLE 状态的时间为尾时间.当网络请求出现间隔式地发送时,就会导致额外的网络能耗开销.

### 1.2 网络请求合并实例

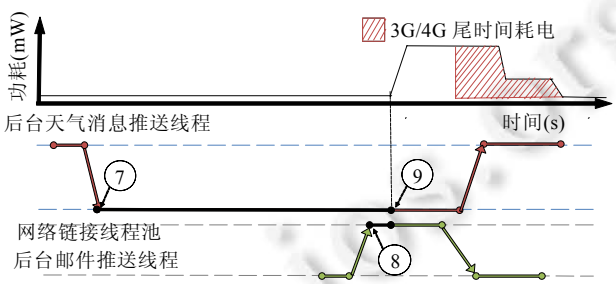
图 2 展示了一个网络请求合并的实例.图 2(a)为合并前的网络请求与无线通信模块耗电,其横轴为时间,上半部分为无线通信模块的功耗;下半部分的虚线为发起这两个网络请求的线程,实线为其控制流.首先,一个后

台的新闻推送线程唤醒了负责发送网络请求的线程(①);该线程被唤醒后,发起了网线请求(②),此时,无线通信模块的功耗也由 IDLE 状态下的低功耗,变为 CELL_DCH 状态下的高功耗;当整个请求完成后,负责发送网线请求的线程将结果返回给新闻推送线程(③),此时虽然无线通信模块没有接收或发送数据,但它仍会保持在高功耗状态,由此开始的无线通信模块耗电被称为尾时间耗电[3],对应为图 2(a)中用的斜线部分;新闻推送线程在收到返回结果后(④),对结果进行处理,并在通知栏上进行提示(⑤).又过了一段时间后,另一个版本更新线程也执行了类似的逻辑(⑥),发送了网络请求.如图 2(a)所示:由于这两个网络请求间隔了几十秒,导致无线通信模块被唤醒了两次,因此也有了对应的两次尾时间,由此导致了额外的网络能耗.

对于安卓应用,有很大一部分后台请求可以被延迟几十秒、甚至两、三分钟,而不会影响用户的体验,例如上述的新闻推送、版本更新推送等.对于这些网络请求,如果在时间维度上进行合并,即,两个请求同时发送而不是间隔了几十秒发送,就可以减少尾时间的网络耗电.图 2(b)为图 2(a)中两个请求进行了合并后的控制流及其无线通信模块耗电情况.首先,负责发送网络请求的线程被新闻推送线程唤醒后,并不直接发送网络请求,而是进入一个等待状态(⑦);一段时间后,另一个网络请求线程被后台更新推送线程唤醒,同时,它也进入一个等待状态(⑧);在等待状态结束后(⑨),这两个线程同时发送网络请求,对应的无线通信模块也只被唤醒一次.如图 2(b)所示,合并后的网络请求的耗电要远小于合并前的网络请求耗电.




(a) 原始网络请求控制流




(b) 合并后网络请求控制流

Fig.2 Motivating example

图 2 网络合并实例

为了实现网络请求合并,需要:(1) 一种网络请求调度机制,即使原本直接发送的网络请求可被延迟发送;(2) 一种网络请求调度算法,即找出可被延迟调度的请求,同时利用调度机制进行延迟发送.

## 2 安卓应用中网络请求合并的方法概述

本文所给的程序转换遵循重构的原则[12],即:对给定的应用程序的内部结构进行修改,而不改变其外部功能.转换包括 3 个基本要素.

(1) 给定安卓应用所具有的原始程序结构,称为源结构(source structure);

(2) 转换后的安卓应用所具有的新的程序结构,称为目标结构(target structure);

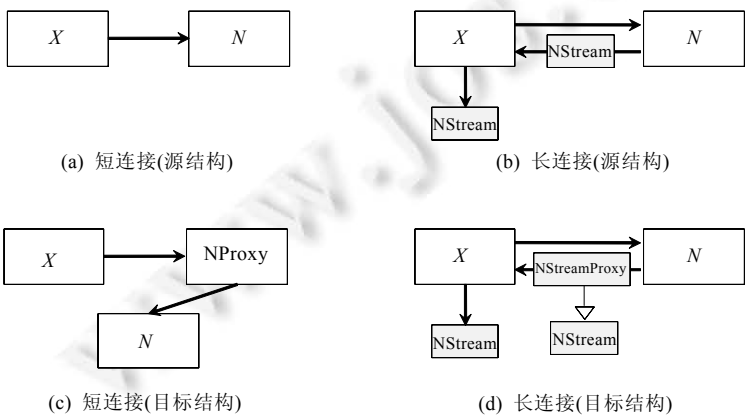
(3) 一系列将源结构变为目标结构的转换操作.

我们首先介绍源结构以及目标结构,并在下一节中给出转换操作概述.

一个给定的安卓应用中的网络连接根据其生命周期的长短可分为:(1) 短连接,即,应用每进行一次网络请求就建立一个 TCP 连接,如图 3(a)所示;(2) 长连接,即,应用多次利用一个 TCP 连接读取、写入数据,如图 3(b)所示.在安卓应用中,典型的短连接包括利用 URLConnection,HttpClient 等类发起网络连接;而利用 Socket 类发起的网络连接则是典型的长连接.

在程序转换过程中,针对短连接和长连接的程序转换略有不同,这是由于这两种连接有着不同的使用模式.在图 3(a)中,类 *N* 为框架层中提供网络请求的相关类,例如 URLConnection 和 HttpClient;应用类 *X* 通过调用类 *N* 的一些方法实现发起网络连接.当应用类 *N* 的方法执行结束,就完成了一个网络请求,应用类 *X* 可通过应用类 *N* 获取到网络请求的结果.为了通过延迟发送网络请求以实现网络请求合并,我们引入了一个类 NProxy,如图 3(c)所示.我们将应用类 *X* 直接调用类 *N* 的方法转换成了经由 NProxy 间接调用类 *N*.而 NProxy 内部在调用类 *N* 前,会利用算法决定直接调用还是延迟调用.

图 3(b)为长连接的调用结构.类 *N* 为框架层中提供网络长连接的相关类,例如 Socket;应用类 *X* 通过调用类 *N* 的一些方法获得一个流对象(NStream);而后,类 *X* 通过该流对象的读写操作实现接收、发送数据.如果采用类似于短连接的程序转换方式,即将类 *X* 直接调用 NStream 的方法转换成调用 NProxy,再由 NProxy 去调用 NStream 的方法,则无法处理以下情况:(1) Stream 对象在安卓应用中很常见,并非所有的 Stream 对象都会接收、发送网络数据,因此,在程序转换的静态分析阶段难以判断某个 Stream 是否为网络相关的 Stream;(2) 一些系统提供的工具类,如 BufferedOutputStream 等会被用来封装一些 Stream 以实现缓冲写等功能.当类 *X* 利用这些工具类对 NStream 进行封装时,它并不直接调用 NStream 的方法,而是由 BufferedOuptutStream 调用.综上所述,针对短连接的程序转换方式不适用于长连接的调用结构.其根本原因在于:NStream 类继承自常见的 Stream 类;而 Stream 类的方法不仅会在应用类 *X* 中被调用,也会在系统提供的工具类中被调用,而程序转换只能针对应用类进行转换.因此,针对长连接的调用结构,我们引入了 NStreamProxy,如图 3(d)所示,它是 NStream 类的一个子类,并且它的外部行为与 NStream 类保持一致.当调用类 *N* 的方法获取得 NStream 时,我们利用 NStreamProxy 对 NStream 进行封装.而 NStream 的涉及网络读/写的方法在 NStreamProxy 类中会被实现为先调用 NProxy,再由 NProxy 去决定直接还是延迟调用 NStream 的对应方法.利用这种实现,不论类 *X* 是直接调用 NStreamProxy 还是利用系统工具类间接调用,都会经由 NProxy 调用 NStream 的相关方法,从而实现对长连接网络请求的延迟发送与合并.




(a) 短连接(源结构) (b) 长连接(源结构)

(c) 短连接(目标结构) (d) 长连接(目标结构)

Fig.3 Source structure of an Android application that sending network request and the target structure it held after transformation

图 3 安卓应用中发起网络连接的源结构与转换后的目标结构

### 2.1 程序转换步骤

在确定了安卓应用的源结构及目标结构以后,我们逐步实施如图 4 所示的程序转换步骤来实现给定安卓应用中的网络请求合并,并保障网络请求合并后的应用网络能耗的节省.

(1) 定位网络相关应用类.

对于一个给定的安卓应用,DelayDroid 通过关键字匹配的方式,自动将其应用类中网络相关的应用类定位出来.这些网络相关的应用类使用了 URLConnection,Socket 等框架层提供的网络相关的类.进一步,DelayDroid 以方法为粒度,排除了那些使用了网络相关的类的方法却并没有调用发送、接收网络请求的方法.通过第 1 步的筛选,DelayDroid 确定了需要进行程序转换的那些类.

(2) 网络相关应用类转换.

在这一步中,我们会把第 1 步中定位出来的应用类从源结构转换为目标结构,使其网络请求具备可调度的能力.一个网络相关的应用类它调用的所有与网络读/写相关的方法都会被替换为 NProxy 中的方法,其获取的长连接对应的 NStream 类会被替换为 NStreamProxy 类.在本文所提出的方法中,一个 NStreamProxy 与其被代理的应用类 NStream 在外部表现上完全一致.即:这两个类继承了同样的父类,实现同样的业务接口,具有同样的方法签名.这样,原本使用了 NStream 类的应用在使用 NStreamProxy 时感觉不到差别.

(3) 应用间通信相关类插桩.

本文提出的方法要实现多个应用间的网络请求合并,就需要利用一种应用间通信机制以实现应用间的网络请求的协同调度.在实现中,需要获取应用运行时的一个特殊上下文对象(Android.content.Context)以利用安卓的广播机制实现应用间通信.通过在可以获取到上下文对象的应用类中进行差桩,并保存该上下文对象,DelayDroid 实现了利用安卓的广播机制实现应用间通信.对该插桩和应用间通信机制的进一步说明将在第 3 节中展开.

(4) 应用打包.DelayDroid 的输入是一个给定的安卓应用.

在经历以上步骤后,DelayDroid 还需将运行时依赖的一些类,如调度算法等,打包进原应用.

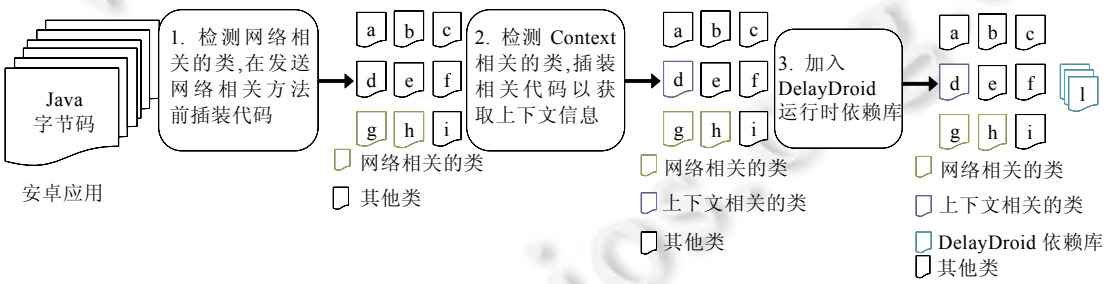



Fig.4 Transformation steps for batching network requests in Android apps

图 4 安卓应用中实现网络请求合并的主要转换步骤

## 3 DelayDroid 的具体实现

DelayDroid 的目标是:在不修改操作系统、不修改源代码、最小化应用开发人员工作量的条件下,实现将应用网络请求合并算法部署到现有的安卓应用中.DelayDroid 的具体实现包含一个针对 dex 格式字节码[9]的程序自动转换工具和一组支撑网络请求调度的运行时库.自动转换工具实现了对安卓应用中的字节码进行转换、打包,并生成优化后的应用.关于该自动转换工具的进一步说明将在第 3.1 节展开.运行时库将网络调度算法与网络调度机制进行解耦,提供了一个默认的网络调度算法,同时提供了一组调度算法接口,开发人员利用这组接口可以方便地、有针对性地实现自己的网络调度算法.关于该运行时库、算法接口及默认调度算法的进一步说明将在第 3.2 节展开.

### 3.1 DelayDroid程序自动转换工具

#### 3.1.1 定位网络相关应用类

在安卓应用开发中,应用开发人员通常利用两类应用编程接口发送网络请求:(1) 安卓开发框架提供的应用编程接口,如 java.net,org.apache.httpclient 等包中的应用编程接口;(2) 第三方提供的网络工具库,例如 Volley[13]和 Retorfit[14].由于这些第三方的工具库也是基于安卓开发框架提供的应用编程接口实现,并且在应用打包时这些工具库的类和其他普通的应用类一起打包起来放在同一个 dex 格式的字节码文件中,因此,DelayDroid 仅需考虑使用了安卓开发框架中提供的网络相关功能的类.本文称应用中这些利用开发框架提供的网络相关的编程接口以实现发送、接收了网络请求的类为网络相关的应用类.

为了定位所有网络相关的应用类,DelayDroid 采用程序静态分析的技术.首先,通过手工收集安卓开发框架中的与发送、接收数据相关的类及其方法,生成了一个配置文件.该配置文件每行包含了:(1) 方法名及其签名;(2) 类名;(3) 是否为静态方法.在具体实现中,该配置文件共包含 16 个类,110 个方法.其次,DelayDroid 根据该配置对应用中字节码的所有应用类的方法中的方法调用进行匹配,若匹配成功,则该方法所在的类会被归类为网络相关的应用类.除此之外,应用开发人员可以通过手工配置该配置文件,以实现识别更多的网络相关类.

#### 3.1.2 网络相关应用类转换

在定位了所有网络相关的应用类之后,DelayDroid 根据其发起的网络连接的类型,是短连接还是长连接进行的程序转换.

对于发起短连接的应用类,其程序转换如图 5 所示.在具体实现中,程序转换是在 dex 字节码级别进行,图 5 显示其对应的 Java 代码,以便于理解.图 5(a)中以“-”开在头的代码为应用类的原始代码,它通过调用 HttpClient 类的 execute 方法,实现发送一个 Http 请求.为了使其具备调度的能力,DelayDroid 将其转换成调用 NProxy 的 execute 方法,即,图 5(a)中以“+”开头的代码.

图 5(b)为 NProxy 中的 execute 方法,在该方法中,除了调用 HttpClient 类的 execute 方法外,还调用了两个额外的方法:DelayController 类的 sleep 方法和 wakeUp 方法.sleep 方法会根据应用指定的算法确定该网络请求所需的延迟时间;而 wakeUp 方法则是唤醒所有在等待的本应用和其他应用的线程.有关于 DelayController 的更多说明会在第 3.2 节展开.

```
1  public class BackgroundWorker  extends  Thread {
2      public void run(){
3          //…
4          sendNetworkRequest();
5      }
6      private void sendNetworkRequest(){
7          //…prepare argument in the request
8          - HttpResponse  r = httpClient.execute(arg0);
9          + HttpResponse  r = NetworkStub.execute(httpClient,arg0);
10          //…
11     }
12}
```

(a) 短连接程序转换示例

```
1 public class NProxy {
2     public static HttpResponse execute(HttpClient  c,
3                                        HttpUriRequest  q){
4         DelayController.sleep();
5         HttpResponse  r = c.execute(q);
6         DelayController.wakeUp();
7         return r;
8     }
9 //…
10}
```

(b) NProxy中的execute方法

Fig.5 Transforming short-lived network call

图 5 短连接程序转换示例

对于发起长连接的应用类,它调用网络相关方法后,返回结果为流对象.利用这个流对象,应用类可以多次地读/写数据.而每次读/写都会对应着网络数据的发送、接收.例如,Socket 类的 getOutputStream 方法会返回一个 OutputStream 对象.当该 OutputStream 对象的 write 方法被调用时,相应的数据就会由本地传输至服务器.对于这种类型的应用类,按照短连接的转换方式进行转换会遇到两个问题:一方面,流对象在应用类中大量使用,由于面向对象编程中的泛化概念在安卓编程中广泛使用,静态分析难以判断一个 write 方法调用是否源自于 Socket 类的 getOutputStream 对象;另一方面,如果开发人员利用框架层编程接口,如 BufferedOutputStream,对这种网络相关的流对象进行封装,由于这些框架层的类的字节码并不存在于应用安装包中,因此,针对短连接的转换方式无法有效地对这些类进行转换.

为了解决以上两个问题,DelayDroid 将原结构转换成一种符合代理模式的目标结构.其程序转换如图 6 所示.类似于图 5,为了便于理解,图 6 展示的代码为 dex 字节码对应的 Java 代码.图 6(a)中,以"-"为开头的代码为原始代码,它通过调用 Socket 类的 getOutputStream 方法获取了一个流对象.DelayDroid 会将该方法调用转换为 NStreamProxy 类中一个同名的方法调用(以"+"为开头的代码).如图 6(b)所示:NStreamProxy 中对应的那个方法并不会直接返回原本的流对象,而会对它进行进一步封装,返回一个 DelayedOutputStream 对象.如图 6(c)所示:DelayedOutputStream 是原本的流对象的代理,它重载了原本流对象中涉及网络操作的方法,例如 write 方法,通过增加 DelayController 的 sleep 与 wakeUp 方法调用,实现了对长连接类型的网络请求的调度.

```
1 public class LongLivedWorker extends Thread {
2     BufferedOutputStream  bos;
3     public void LongLivedWorker(SocketAddressaddr){
4         //…
5         bos = new BufferedOutputStream(
6                     - socket.getOutputStream()
7                     + NetworkStub.getOutputStream(socket)
8                 );
9     }
10    public void sendData(String data){
11        bos.write(data);
12    }
13    //…
14}
```

(a) 长连接程序转换示例

```
1 public class NStreamProxy{
2     public static getOutputStream(Socket s){
3         DelayController.sleep();
4         OutputStream out = s.getOutputStream();
5         out = new DelayedOutputStream(out);
6         DelayController.wakeUp();
7         return out;
8     }
9     //…
10}
```

(b) NStreamProxy中的getOutputStream方法

```
1 public class DelayedOutputStream extends OutputStream {
2     private OutputStream out;
3     public DelayedOutputStream(OutputStream out) {
4         this.out = out;
5     }
6     public void write(int paramInt) throws IOException {
7         DelayController.sleep();
8         this.out.write(paramInt);
9         DelayController.wakeUp();
10    }
11    //…
12}
```

(c) DelayedOutputStream中的write方法

Fig.6 Transforming long-lived network call

图 6 长连接程序转换示例

3.1.3 应用间通信相关类插桩

经以上两步的程序转换,应用已经具备了对网络请求调度的能力.DelayDroid 的目标是调度多个应用间的网络请求,以合并更多的网络请求,达到减少整体网络能耗的目标.具体而言,实现应用间的消息通信需要两个基本操作:1) 发送消息,通知其他应用当前可以发送网络请求;2) 接收消息,唤醒当前于等待状态的线程.在此,主要的挑战在于如何在没有源代码的情况下,将这两个消息操作实现到现有的安卓应用中.

DelayDroid 利用安卓的一种进程间通信机制——广播机制[15],通过定义一个新的广播消息,Wakeup 消息,并且在应用中注册相应的消息接收器,实现了多个应用间的消息通信.然而对于出于安全的考量,普通的应用类并不能直接发送广播消息,也不能注册相应的消息接收器.它们需要利用一个应用的 Context 对象[16]实现才能实现以上操作.DelayDroid 通过应用字节码转换,实现获取应用的 Context 对象.

根据安卓开发人员文档,Context 对象可以通过 ContextWrapper 对象获取.而 ContextWrapper 类包含 31 个子类,包括了安卓应用中重要的两个类:Activity 和 Server.当应用启动时,一个继承了 Activity 的应用类会被实例化,并且它的 onCreate 方法会被调用.在 onCreate 方法中,“this”关键字指向的对象,即为一个 Activity 对象,同时也是一个 ContextWrapper 对象.据此,DelayDroid 进行了以下程序转换步骤:1) 构造出每个应用类的继承链[17],根据对比其继承链中是否包含 ContextWrapper 对象判断该类是否为需要进行应用程序转换的类;2) 若该应用类继承了 ContextWrapper,则对其 onCreate 方法进行如图 7 所示的插桩.以%开头的代码为插桩的代码,其中,ContextService 为 DelayDroid 运行时库的类,通过以“this”作为参数调用它的初始化方法,实现获取应用全局的 Context 对象,进而可以进行注册自定义的 Wakeup 消息接收器,实现多应用间的消息通信.

```
1 public class SimpleActivity extends Activity {
2     public void onCreate(Bundle b){
3         + ContextService.initContextService(this);
4         //…
5     }
6 }
```

Fig.7 Instrumenting to get Context object

图 7 通过程序转换获取 Context 对象示例

3.1.4 运行时支撑库注入与应用打包

经过以上 3 步的程序分析和程序转换,还需给原应用注入 DelayDroid 的运行时库.DelayDroid 运行时库实现的主要功能包括应用间通信机制和网络请求调度算法.关于运行时库的进一步说明将在第 3.2 节展开.最后,利用签名工具给打包好的 apk 文件重新签名,即完成整个程序转换流程.

### 3.2 DelayDroid运行时

经过程序转换后的应用,其运行时体系结构如图 8 所示.运行时,应用类可分为 4 种:1) 被转换后的网络相关应用类;2) 被插桩后的上下文相关应用类;3) DelayDroid 运行时库中的类;4) 其他无关类.其中,DelayDroid 运行时库中的类包括上下文服务(ContextService)和调度控制器(DelayController).图 8 中的箭头表示控制流.当应用启动时,应用类中继承了 Activity 的那些类会被实例化,随后,它们的 onCreate 方法也会被调用.如第 3.1.3 节所述,ContextService 类会在这些 onCreate 方法被调用时进行初始化,并利用获取的 Context 对象以实现应用间的通信.随后,ContextService 会初始化 DelayController,即,根据配置文件选择一个网络请求调度算法.当网络相关的应用类准备发起网络请求时,会先调用 DelayController 相关的方法,根据当前的网络请求调度算法决定立即发送请求还是延迟发送.

除此之外,为了支持多个应用间的网络请求的合并,DelayDroid 基于安卓的广播机制,定义了新的消息——唤醒消息.ContextService 会在初始化时自动注册唤醒消息的监听器.当有来自其他应用的唤醒消息时,ContextService 会调用 DelayController 的相应方法,由 DelayController 回调算法中定义的回调函数唤醒应用中等待的网络请求.

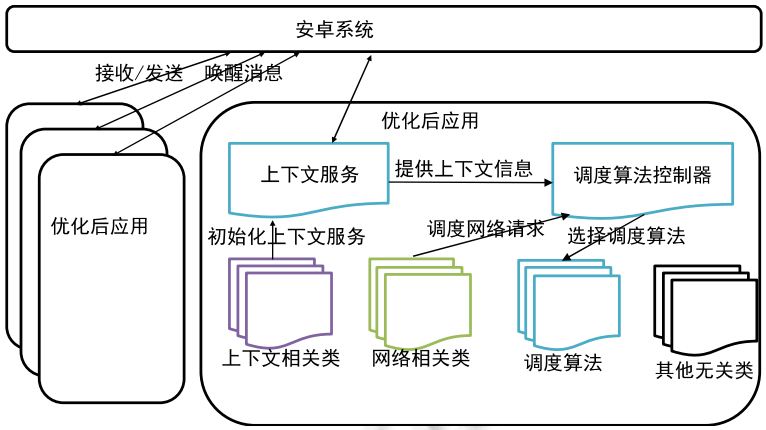



Fig.8 Runtime architecture

图 8 运行时体系结构

在具体实现中,应用可以采取不同的网络请求调度算法.因此,DelayDroid 将调度算法抽象成了接口,并提供给应用开发人员.应用开发人员针对自己的应用可以实现有针对性调度算法.关于调度算法接口将在第 3.2.1 节进一步展开说明.除此之外,DelayDroid 还提供了一种基于屏幕状态的调度算法.在默认情况下,会使用该算法优化应用的网络耗电.该算法将在第 3.2.2 节展开说明.

3.2.1 调度算法接口

调度算法的核心思想是:通过延迟一部分网络请求的发送,使其与其他网络请求一起发送,以减少唤醒蜂窝模块的次数,节省手机能耗.一个好的调度算法可以最小化蜂窝模块的唤醒次数,并且在蜂窝模块唤醒时最大程度地发送网络请求.不同的调度算法的区别在于有不同的策略决定网络请求延迟,而其算法的流程都有两个关键的部分:(1) 决定每个网络请求的延迟大小;(2) 接收到唤醒消息时,应当唤醒其他正在等待的网络请求.

因此,调度算法可抽象为以下接口.

1) start:在一个网络请求发起前,start 方法会被 DelayController 的 sleep 方法调用,调度算法通过实现该方法决定当前请求最多可以延迟多久后发送;
2) issue:当应用接收到唤醒消息时,issue 方法会被调用,应用开发人员可以通过实现这个方法唤醒被阻塞的线程,以减少线程的阻塞时间;
3) finish:当一个网络请求结束时,finish 方法会被 DelayController 的 wakeUp 方法调用.开发人员可以在该方法中利用 ContextService 提供的接口发送唤醒消息,以唤醒其他应用中等待的网络请求.

3.2.2 调度算法示例

利用 DelayDroid,可以将不同的调度算法部署至移动应用.例如:根据是否为当前进程进行优先级排序,对于当前用户交互相关的请求快速发送;对于后台应用的网络请求进行延迟合并.由于合并算法并非本文主要关注部分,因此,基于 Huang[4]提出的假设,我们提出一个调度算法示例:当手机屏幕关闭时,用户很可能并不使用手机,因此,此时的网络请求更可能被延迟调度.为了最小化对用户的影响,我们的算法仅在手机屏幕关闭时对网络请求进行调度.该算法的策略如下.

1) 如果屏幕开启,则不对网络请求进行调度;
2) 如果屏幕关闭时,网络请求将会根据用户设置的等待周期,等待与其他网络请求合并的机会.等待周期为用户可配置的一个值,它表示一个网络请求最长的等待时间,例如 120s;
3) 应用中任意网络请求被发送时,其他被延迟的网络请求也会立即被发送.

图 9 为该算法中“start”“issue”和“finish”这 3 种方法的流程图.如图 9(a)所示.

当 start 方法被调用时,它首先会判断当前屏幕状态(①);如果为开启状态,则直接返回(②);否则,进一步判断

(③);如果在近 5s 内有网络请求被发送,则直接返回(④);否则,将该线程挂起,并记录该线程应该被唤醒的时间(⑤);在线程等待过程中,如果接收到其他应用的唤醒消息(⑥)、达到唤醒时间(⑦)或是屏幕被开启了(⑧),都会将线程唤醒(⑨),使其返回并发送网络请求(⑩).在完成网络请求发送后,还会调用 finish 方法.在实现中,为了避免发送大量的唤醒消息,finish 并不会直接发送唤醒消息.如图 9(b)所示:当 finish 被调用时,它首先会更新最近请求时间(①),然后进行判断(②),如果在一个等待周期内没有发送、或接收过唤醒消息(③),则发送一个唤醒消息,然后返回(⑤);否则直接返回(④).根据 finish 的实现,系统中的唤醒消息发送的最高频率为每个等待周期一条.这种实现可以减少该合并算法对手机中央处理单元(CPU)的资源使用.

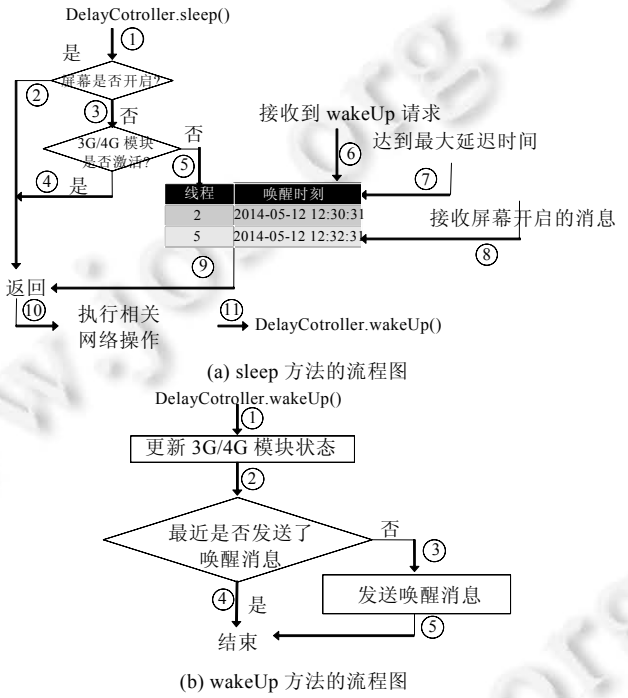



Fig.9 Batching algorithm example

图 9 合并算法示例

## 4 实 验

在本节,我们要验证:(1) DelayDroid 程序自动转换工具的可用性;(2) 经 DelayDroid 工具转换的应用的网络能耗节省.

### 4.1 DelayDroid的可用性

为了验证 DelayDroid 的可用性,我们从谷歌应用商店和豌豆荚应用商店中的天气、邮件、新闻、社交等 6 个类别中,选择了国内外开发的应用各 6 个,共计 12 个应用作为本节的研究对象.见表 1,这些应用的用户量高达几十万至数百万,具有一定代表性.我们使用一台普通的笔记本对这些应用进行自动转换,其中央处理器为英特尔 i5-3247U(1.8GHz-2.6GHz),内存为 4GB,存储为 128GB 固态硬盘.

DelayDroid 工具处理以上 12 个应用所花的时间由 4s~15s 不等,如图 10 所示.主要耗时在于分析和程序转换.由于 DelayDroid 工具仅需应用开发者在发布应用前进行使用,应该说,DelayDroid 处理时间是相当快的.经过 DelayDroid 工具处理后的 apk 仅会比原应用大 64KB 左右,主要来源于添加了 DelayDroid 运行时库.

在静态分析阶段,DelayDroid 会根据配置文件定位网络相关 API.我们将所有这些网络相关 API(共计 110

个)按照其发送的网络请求类型和其所在的包分成了 5 类,见表 2.

**Table 1** 12 off-the-shelf apps from google app store and wandoujia app store

**表 1** 谷歌与豌豆荚应用商店中选取的 12 个应用及其安装量

| 谷歌应用商店中选取的应用 | | | 豌豆荚应用商店中选取的应用 | | |
|---|---|---|---|---|---|
| 应用名 | 类别 | 安装量 | 应用名 | 类别 | 安装量 |
| Weather-Channel | 天气 | 50 000 000+ | 360 天气 | 天气 | 5 000 000+ |
| FoxNews | 新闻 | 5 000 000+ | 今日头条 | 新闻 | 50 000 000+ |
| Pinterest | 社交 | 100 000 000+ | 人人 | 社交 | 10 000 000+ |
| Gmail | 邮件 | 1 000 000 000+ | QQ 邮箱 | 邮件 | 10 000 000+ |
| Amazon | 购物 | 50 000 000+ | 一号店 | 购物 | 5 000 000+ |
| Pandora | 音乐 | 100 000 000+ | 豆瓣电台 | 音乐 | 7 000 000+ |

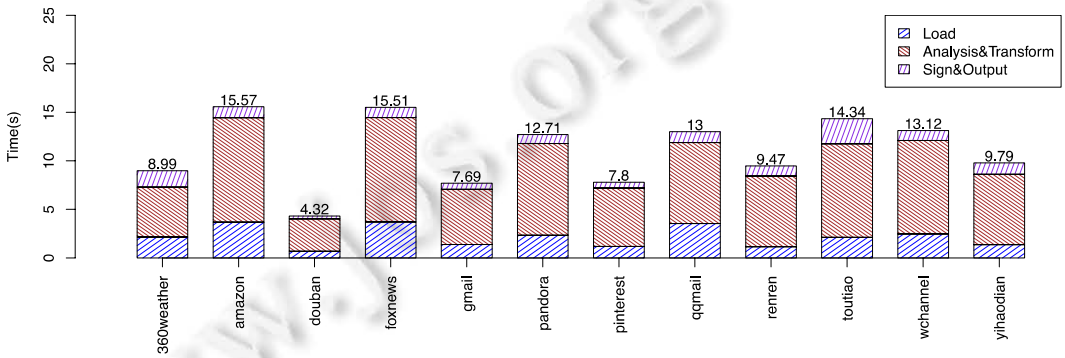



Fig.10 The time consumed in tranformation using DelayDroid

图 10 DelayDroid 工具程序转换时间

**Table 2** Classification of the network calls currently handled by DelayDroid

**表 2** DelayDroid 可处理的网络相关 API 分类

| 类别 | 说明 |
|---|---|
| HttpWebkit | 位于 android.webkit 包中,发送 HTTP 请求的方法 |
| HttpApache | 位于 org.apache 包中,发送 HTTP 请求的方法 |
| HttpJavaNet | 位于 java.net 包中,发送 HTTP 请求的方法 |
| TCPSocket | 发送 TCP 包的方法,例如 Socket 类中的“send”方法 |
| UDPSocket | 发送 UDP 包的方法,例如 DatagramSocket 中的“send”方法 |

DelayDroid 对以上 12 个应用的静态分析结果如图 11 所示.

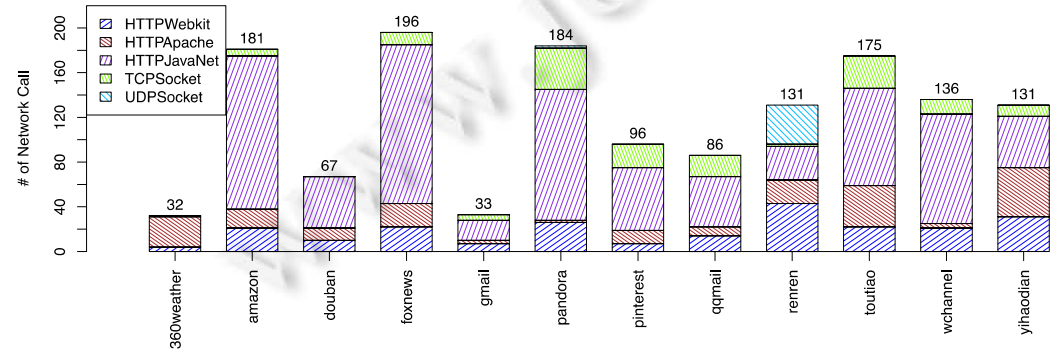



Fig.11 Number of network calls in different categories

图 11 应用网络相关方法数量及分类

首先,HttpWebkit 中的 API 在这 12 个应用中使用最多.这是因为 HttpWebkit 中的 API 常用于在应用中嵌入网页.这些应用利用这些 API 直接嵌入新闻页面、广告页面等,可以高效地复用以前的代码.其次,HttpApache 和 HttpJavaNet 两个类别的 API 在这 12 个应用中使用较多.这是因为安卓应用一般都用 Http 协议调用服务器的服务,而在安卓开发文档中,推荐使用属于以上两个类别的 API 发送 Http 请求.同时,这两类 API 也常为第三方网络库使用,例如 Volly 和 Retrofit.TCPSocket 类的 API 提供较底层的、可靠的客户端-服务器通信机制.邮件客户端,如 Gmail 邮箱,利用这类 API 实现 IMAP 协议和 POP3 协议.UDPSocket 提供的是不可靠的数据包通信机制,仅在人人和 Pandora 应用中出现.

DelayDroid 程序自动转换工具会使网络相关方法的调用变得更复杂——每个方法调用会被封装成一个新的方法,并包含了两个额外的方法调用.然而,发起一个网络请求的开销要远大于简单的方法调用,因此,这些额外的方法调用的计算资源开销几乎可以忽略.

### 4.2 DelayDroid的网络能耗节省

DelayDroid 节省网络能耗的原理是通过延迟部分网络请求的发送,使之与其他网络请求同时发送,以减少蜂窝网络模块的唤醒次数,降低尾时间耗电.在一个智能手机中,更多的应用一起运行,往往意味着会发起更多的网络请求,同时也意味着有更多的减少蜂窝网络的唤醒机会.

在第 4.1 节的 12 个应用中,可分为天气、邮件、新闻、购物、社交这 5 类,由于在网络推送的使用上,购物类应用的消息推送与新闻的消息推送较为相似,故我们选择了网络使用较频繁,且实时性要求较低的天气、邮件、新闻这 3 类应用的 3 个代表应用:天气应用(360 天气)、邮件应用(QQ 邮件)和新闻应用(今日头条).而社交类应用用户对消息延迟的敏感性较高,且用户交互与网络请求延迟的讨论不在本文范围内,故下述实验仅研究 360 天气、QQ 邮件和今日头条这 3 个应用.这些应用在屏幕关闭时发送的网络请求对延迟要求不敏感.与即时通信软件不同,在屏幕关闭时,天气和新闻推送消息推迟两三分钟对用户而言都是可接受的.这 3 个应用的设置见表 3.在下面的实验中,我们会采用第 3.2.2 节中的默认算法,对 DelayDroid 在单个应用下和多应用下的网络耗电进行评估.我们采用红米 2 手机,网络连接为移动 4G.手机运行精减版 CynaogenMod 10[18]系统,安卓版本为 4.4.4.相比于原生的米 UI 系统,这个系统自带的系统服务较少,便于控制后台运行的应用数量.在测量能耗方面,我们采用网络跟踪文件模拟计算的方法[19]计算手机的网络耗电.模拟计算网络能耗所使用的参数见表 4.本文从单个应用网络能耗、多个应用网络能耗两个方面对 DelayDroid 进行评估.

**Table 3** App configurations for runtime experiements

**表 3** 实验中的应用设置

| 应用 | 配置 |
|---|---|
| 360 天气 | 仅添加一个地点,北京 |
| QQ 邮件 | 添加一个测试邮箱 |
| 搜狐新闻 | 默认设置 |

**Table 4** Parameters of network energy model

**表 4** 网络耗电模型

| 参数 | 值 |
|---|---|
| CELL_DCH 功耗 | 1 950.0mW |
| CELL_FACH 功耗 | 1 060.0mW |
| 尾时间 | 11 576.0ms |

#### 4.2.1 单个应用的网络能耗

我们将上述的天气、邮件和新闻应用依次安装在测试手机中,针对每个应用,我们的算法采用了 4 种等待周期:0s(即不等待)、30s、60s 和 120s,每个等待周期都进行 24 小时的测试时间.在测试过程中,手机屏幕保持关闭.我们在算法中增加记录网络请求的延迟时间、发送时间、结束时间,从而获取网络跟踪文件.利用文献[19]的方式,模拟计算每个应用的网络耗电.

实验结果如图 12 所示.图中的折线部分表示不同等待周期的网络能耗;柱状部分表示网络模块的唤醒次数.我们将每次唤醒与上一次唤醒的时间间隔进行分类,分为间隔小于 30s、30s~60s、60s~120s 和 120s 这 4 类.例如,如果本次唤醒与上次唤醒间隔小于 30s,则我们的算法在等待周期设置为 30s 以上时可合并这两次唤醒,实现节省网络能耗的目的.由图 12 可发现:对于天气应用和邮件应用,它们都是间隔式地发送网络请求,周期大约为 1 小时一次和 10 分钟一次.因此,当这两个应用分别运行时,DelayDroid 并不能合并网络请求.当等待周期为 120s 时,这两个应用的唤醒次数都有略微的下降(由 25 次降至 24 次;由 303 次降至 287 次),这是因为即便是没有合并网络请求,由于每个网络请求都额外等待了两分钟,导致它们的发送周期增长了,即唤醒频率下降了.对于搜狐新闻,由于该应用的推送逻辑更为复杂,因此网络唤醒次数更多.从等待周期为 0s 的数据上看,一天之内,唤醒次数为 549 次,其中,有大约 180 次是唤醒间隔小于 30s 的,这表明有相当一部分网络请求都是可合并的.对比搜狐新闻中等待周期为 30s 与等待周期为 0s 的数据可以发现:在等待周期为 30s 时,唤醒次数降为 357 次,并且两次唤醒间隔时间小于 30s 的次数明显减少;当等待周期进一步增加到 120s 时,唤醒次数为 239 次,比等待周期为 0s 的减少了近 50%.

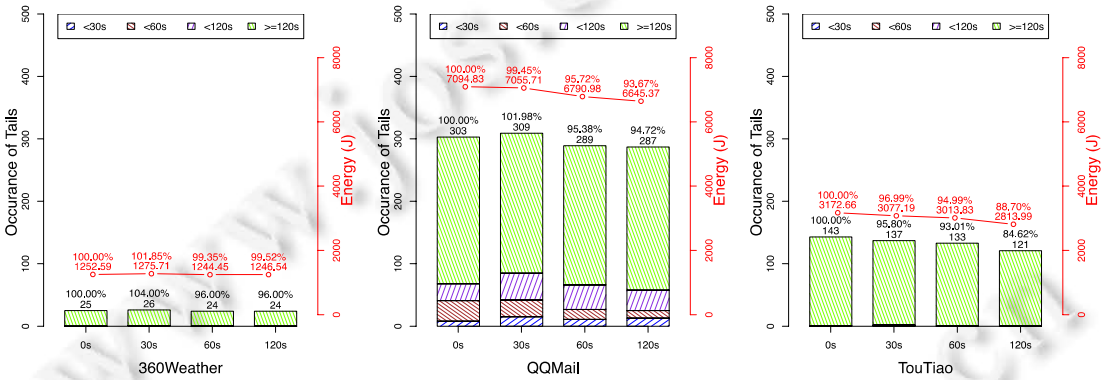



Fig.12 The radio energy consumption in each individual app and the occurrence of tails

图 12 单应用网络能耗与尾时间出现次数

图 12 的折线部分为每个应用在不同等待周期下的网络能耗.对比等待周期 0s 和 120s,天气应用的网络能耗无明显减少;邮件应用与新闻应用分别节省了 6.3%和 11.3%.这些网络节省来源于两个方面:(1) 增加网络请求的延迟导致的周期性网络请求的发送周期延长,由此引起的网络请求的数量减少;(2) 合并网络请求,减少蜂窝模块的唤醒次数.图 13 为 3 个应用在不同等待周期下的网络请求个数,对比图 12 的能耗数据可以发现:对于这 3 个应用而言,难以实现单个应用内的请求合并以优化网络能耗.

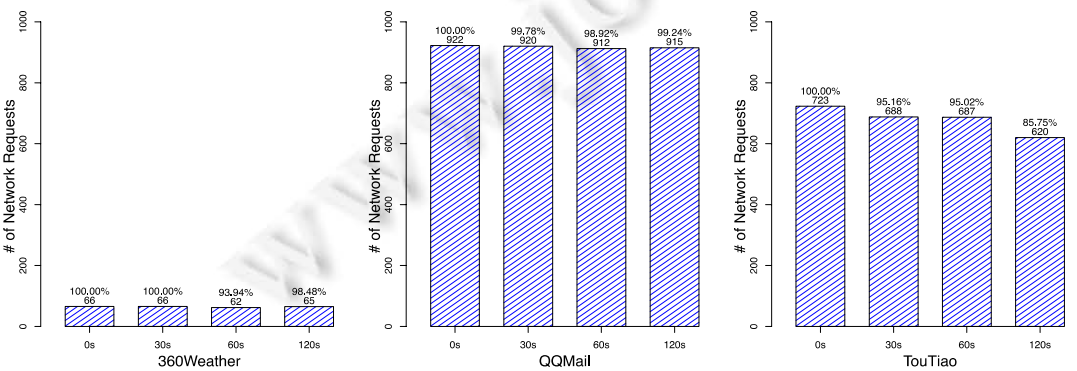


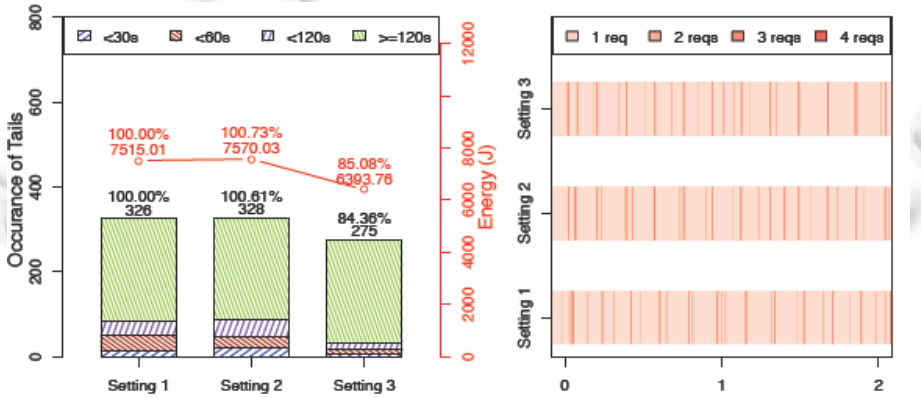

Fig.13 Number of network requests in different apps with different batching window size

图 13 不同应用在不同周期下的网络请求个数

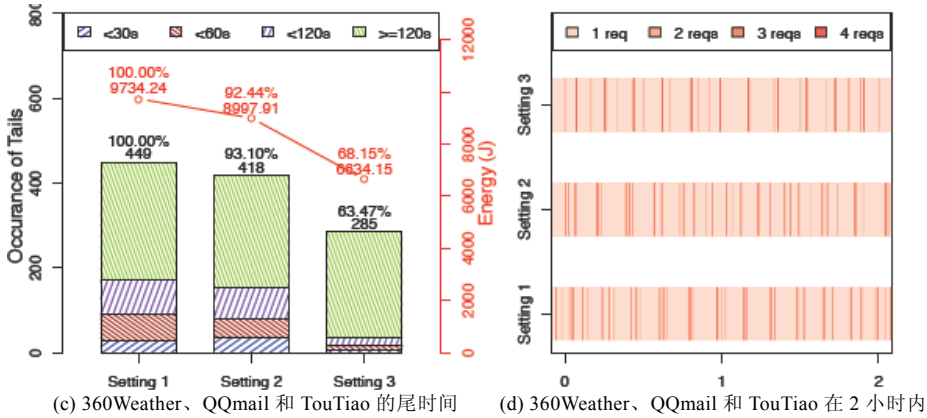
4.2.2 多个应用的网络能耗

对于以上 3 个应用,由于应用开发者均对自己应用的网络耗电进行了优化,因此难以实现单个应用内的请求合并.而当天气、邮件和新闻这些应用运行在一起时,则有一定机会合并应用间的网络请求,从而实现节省手机的网络能耗.在本节中,我们评估多个应用一起运行的节能效果.首先,测试两组应用:(1) 天气应用与邮件应用;(2) 天气应用、邮件应用与新闻应用.针对每组应用,我们采用以下设置:(1) 等待周期为 0;(2) 不启用进程通信机制,等待周期为 120s;(3) 启用进程通信机制,等待周期为 120s.

实验结果如图 14 所示.图 14(a)为天气应用与邮件应用同时运行时的网络能耗与唤醒次数.对比设置 1 与设置 2,网络能耗节省效果类似于单个应用运行时的情况,并不能节省网络能耗.而当启用了进程通信机制时(设置 3),应用间的网络请求合并被启用,相比于没有启用能够节省 15%的网络能耗.相应地,对比设置 1 与设置 3,其唤醒次数也减少了 15%.我们取了 2 小时的网络跟踪文件,如图 14(b)所示.图 14(b)的横轴为时间:0~2 小时;纵轴为 3 种设置.每个网络请求在图中为矩形,由于网络请求持续时间很短,因此在图中表现为线条.越深的表示越多网络请求同时发送,也就表示其合并程度更高.从图 14(b)中可以直观地发现:通过启用进程通信机制,可以有效地合并应用间的网络请求.类似地,图 14(c)所示为 3 个应用同时运行的网络能耗与唤醒次数.对比设置 1 与设置 2,其网络能耗仅减少了 8%,唤醒次数由 449 减少为 418 次,这主要来源于网络请求的减少,与前一节结果相符.而当我们启用了多应用之间的通信机制,,相比设置 1,其网络能耗减少了 32%,唤醒次数降低了 37%.以上结果表明:利用 DelayDroid 的多应用请求合并机制,能够减少智能手机的网络能耗.




(a) 360Weather 和 QQmail 的尾时间出现次数和能耗 (b) 360Weather 和 QQmail 在 2 小时内的网络请求




(c) 360Weather、QQmail 和 TouTiao 的尾时间出现次数和能耗 (d) 360Weather、QQmail 和 TouTiao 在 2 小时内的网络请求

Fig.14 The radio energy consumption in groups of apps and the occurrence of tails

图 14 多应用网络能耗与尾时间出现次数

## 5 相关工作

有许多已有的相关工作是优化 3G/4G 网络能耗,这些工作可以被分为以下 3 类.

- 合并算法与策略

TailEnder[3]首次提出了 3G 网络下,由于尾时间导致的能耗浪费的问题.除此之外,TailEnder 还提出了通过调度网络请求,对一些可以延迟发送的请求增加一定延迟,以合并多个网络请求,从而节省 3G 网络下的耗电.从模拟的实验结果看,通过增加延迟可以节省高达 50%的网络能耗.TailEnder 为 3G 网络耗电的第一篇相关文献,由此开展了一系列的后续研究工作.

Huang[19]对移动应用中那种定期的网络请求进行了研究,它发现:这种请求的特点是传输量小、时间持续短,一般是由于消息推送、轮询、广告推送、用户行为追踪等.该文为第一篇对应用后台的网络请求耗电做了分析的论文.最后,对现有的一些优化策略(如上文中的 TailEnder 调度策略,还有 3G 快速休眠等)做了比较.最后发现:在真实的一些应用中,如 Pandora、新闻等,TailEnder 策略能节省 20%左右的能耗.

后续还有工作[4]分析了移动设备在屏幕开启和屏幕关闭时,其网络请求的特点.经过对真实的应用数据的分析,即便是在屏幕关闭时,还是有不少数量的网络请求,并且当多个应用一起运行时,3G 下的网络耗电有很大一部分浪费在了所谓的尾时间上.根据屏幕这个策略仅在屏幕关闭时才生效.当屏幕暗时,网络请求不会立即被发送,而是进行一定的延迟.这个延迟由提前设置,如 30s、60s 等.在这个网络请求被延迟发送的这段时间里,如果还有网络请求需要发送的话,则会在时间维度上合并,一起发送.

以上都是通过从真实应用中获取的网络跟踪文件进行评估.DelayDroid 可以进一步地将这些工作提出的策略部署到真实应用中,可以提高评估的可靠性.

- 快速休眠

快速休眠是另一种节省 3G/4G 网络能耗的技术.其核心是修改 RRC 状态机的状态转移时间,或是给应用开发人员提供修改这个状态机的一些接口[20].TOP[5]提供了一个简单的 API,应用开发人员利用这个 API 可以将无线资源控制模块由唤醒状态直接转移到休眠状态.

RadioJockey[21]则是利用程序执行的跟踪文件来预测网络请求的时机.当一个网络请求结束时,如果预测了后面没有网络请求,则直接利用快速休眠技术将无线资源控制模块设置为休眠状态.类似于快速休眠,还有一些工作是根据运行时的网络收、发状态,动态修改 RRC 状态机的转移时间[22,23].

这些工作与 DelayDroid 是互补的.结合这些工作可以节省更多的网络能耗.

- 算法部署与应用开发

在简化网络合并算法的部署方面,相关工作可分为应用层、框架层和系统层这 3 个层次.在应用层,Xu[24]以邮件应用为研究目标,分析了邮件同步的网络发送特点,提出了 5 种优化邮件应用的方法,并取得了明显的优化效果.这种优化可以达到对于单个应用省电最佳的效果,然而对于智能手机而言,蜂窝网络模块是多个应用共享的.这种优化难以实施到其他类型的应用上.在框架层,APE[25]提出了一种基于注解的安卓应用开发框架.利用 APE 提供的注解,开发人员可以简化低能耗安卓应用的开发.类似于 Xu[24],这个技术也是针对单个应用的优化,开发人员即便是手工指定数百个网络相关方法,难以实现全局最优.Li[26]对移动应用的 Http 请求进行了合并优化,然而没有实现应用间的网络请求合并.Vergara[27]则是在系统层利用 Linux 内核的 Netfilter 框架提出了一种网络包调度算法,以优化智能手机的网络能耗.在系统级别上的网络调度可以实现合并所有应用间的网络请求,然而由于缺少应用相关的信息,难以达到应用层和框架层的优化效果.DelayDroid 通过提供框架层的编程接口和系统层的应用间网络请求合并机制,既可以实现系统级别的网络调度,也可以实现框架层的优化效果.

在应用开发方面,Ravindranath 提出了一个针对 Windows 手机的程序自动转换框架 Procrastinator[28].它通过算法优化应用预取网络资源,以节省手机的网络流量.DelayDroid 与 Procrastinator 是互补的,只是优化的目标不同.Dpartner[29]则是针对 Java 字节码的一个程序自动转换框架,它提出了一种通过自动程序转换来实现 Java 应用中计算按需远程执行的方法,实现计算的按需远程执行,达到保障性能和提高资源利用率.

与以上工作相比,本文提出的方法和 DelayDroid 的实现具有两个最大特色.

1) 转换过程对开发者保持透明.DelayDroid 不需要开发者修改其源代码,而是通过自动分析完成转换工作,极大地降低了应用开发者的工作量;
2) 转换后的应用支持合并多个应用间的网络请求.DelayDroid 通过提供算法接口,支持开发人员提供应用特定的合并算法的同时,还提供应用间通信机制,实现支持多应用的网络请求调度.

## 6 结束语

应用后台网络请求会极大地影响智能手机的待机时间.本文给出了一种通过应用字节码自动程序转换来实现现有的安卓应用中网络请求延迟调度的方法.本文介绍了将安卓应用转换成支持后台网络请求调度的应用的技术挑战、处理机制以及 DelayDroid 转换系统.与已有的工作相比,DelayDroid 有两大特色:一是程序转换自动执行;二是转换后的应用可支持多应用的网络请求调度,该调度机制可以降低安卓应用的待机耗电.此外,DelayDroid 被设计为可对只有 dex 字节码的安卓应用进行转换,更具实用性.

**致谢** 在此,我们向对本文的工作给予支持和建议的同行,尤其是向北京大学信息科学技术学院软件研究所系统软件与中间件小组的老师和同学表示感谢.

**References**:

[1] Yang FQ, Mei H, Lü J, Jin Z. Some discussion on the development of software technology. Acta Electronica Sinica, 2002,30(12A): 1901−1906 (in Chinese with English abstract). [doi: 10.3321/j.issn:0372-2112.2002.z1.001]

[2] Yang FQ, Lü J, Mei H. Technical framework for internetware: An architecture centric approach. Science in China Series E: Technological Sciences, 2008,38(6):818−828 (in Chinese). [doi: 10.1007/s11432-008-0051-z]

[3] Balasubramanian N, Balasubramanian A, Venkataramani A. Energy consumption in mobile phones: A measurement study and implications for network applications. In: Proc. of the 9th ACM SIGCOMM Conf. on Internet Measurement Conf. ACM Press, 2009. 280−293. [doi: 10.1145/1644893.1644927]

[4] Huang J, Qian F, Mao Z M, Sen S, Spatscheck O. Screen-Off traffic characterization and optimization in 3G/4G networks. In: Proc. of the ACM Conf. on Internet Measurement Conf. 2012. 46−62. [doi: 10.1145/2398776.2398813]

[5] Qian F, Wang Z, Gerber A, Mao Z M, Sen S, Spatscheck O. TOP: Tail optimization protocol for cellular radio resource allocation. In: Proc. of the IEEE ICNP. 2010. 285−294. [doi: 10.1109/ICNP.2010.5762777]

[6] Chen W, Wei J, Huang T. W$^4$H: An analytical framework for software deployment technologies. Ruan Jian Xue Bao/Journal of Software, 2012,23(7):1669−1687 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4105.htm [doi: 10.3724/SP.J.1001.2012.04105]

[7] PeriodicTask. https://developers.google.com/android/reference/com/google/android/gms/gcm/PeriodicTask

[8] Android platform versions. https://developer.android.com/about/dashboards/index.html

[9] Dex bytecode. https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html

[10] 3GPP. http://www.3gpp.org/specifications

[11] 4GPP. http://4gpp-project.net/

[12] Beck BK, Fowler M. Bad smells in code. In: Proc. of the Refactoring—Improving the Design of Existing Code. 2010.

[13] Volley. https://developer.android.com/training/volley/index.html

[14] Retorfit. https://github.com/square/retrofit

[15] https://developer.android.com/reference/android/content/BroadcastReceiver.html

[16] https://developer.android.com/reference/android/content/Context.html

[17] https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

[18] CynaogenMod. https://github.com/CyanogenMod

[19] Huang J, Qian F, Gerber A, Mao ZM, Sen S, Spatscheck O. A close examination of performance and power characteristics of 4G LTE networks. In: Proc. of the Int'l Conf. on Mobile Systems, Applications, and Services. ACM Press, 2015. 225−238. [doi: 10.1145/2307636.2307658]

[20] Chuah M, Luo W, Zhang X. Impacts of inactivity timer values on UMTS system capacity. In: Proc. of the Wireless Communications and NETWORKING Conf. (WCNC 2002). Vol.2. IEEE, 2002. 897−903. [doi: 10.1109/WCNC.2002.993390]

[21] Athivarapu PK, Bhagwan R, Guha S, Navda V, Ramjee R. RadioJockey: Mining program execution to optimize cellular radio usage. In: Proc. of the Int'l Conf. on Mobile Computing and Networking. ACM Press, 2012. 101−112. [doi: 10.1145/2348543.2348559]

[22] Puustinen I, Nurminen JK. The effect of unwanted internet traffic on cellular phone energy consumption. In: Proc. of the 2011 4th IFIP Int'l Conf. on New Technologies, Mobility and Security (NTMS). IEEE, 2011. 1−5. [doi: 10.1109/NTMS.2011.5720613]

[23] Yeh JH, Chen JC, Lee CC. Comparative analysis of energy-saving techniques in 3GPP and 3GPP2 systems. IEEE Trans. on Vehicular Technology, 2009,58(1):432−448. [doi: 10.1109/TVT.2008.923687]

[24] Xu F, Liu Y, Moscibroda T, Chandra R, Jin L, Zhang YG, Li Q. Optimizing background email sync on smartphones. In: Proc. of the Int'l Conf. on Mobile Systems, Applications, and Services. 2013. 55−68. [doi: 10.1145/2462456.2464444]

[25] Nikzad N, Chipara O, Griswold WG. APE: An annotation language and middleware for energy-efficient mobile application development. 2014. 515−526. [doi: 10.1145/2568225.2568288]

[26] Ding L, Lyu YJ, Gui JP, Halfond WGJ. Automated energy optimization of http requests for mobile applic ations. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM Press, 2016. 249−260. [doi: 10.1145/2884781.2884867]

[27] Vergara EJ, Sanjuan J, Nadjm-Tehrani S. Kernel level energy-efficient 3G background traffic shaper for android smartphones. In: Proc. of the Wireless Communications and Mobile Computing Conf. IEEE, 2013. 443−449. [doi: 10.1109/IWCMC.2013.6583599]

[28] Ravindranath L, Agarwal S, Padhye J, Riederer C. Procrastinator: Pacing mobile apps' usage of the network. In: Proc. of the 12th Annual Int'l Conf. on Mobile Systems, Applications, and Services. ACM Press, 2014. 232−244. [doi: 10.1145/2594368.2594387]

[29] Zhang Y, Huang G, Liu XZ, Mei H, Li Y, Yang SX. Approach to supporting on-demand remote execution of the computations in a Java application. Ruan Jian Xue Bao/Journal of Software, 2013,24(8):1713−1730 (in Chinese with English abstraction). http://www.jos.org.cn/1000-9825/4344.htm [doi: 10.3724/SP.J.1001.2013.04344]

**附中文参考文献**:

[1] 杨芙清,梅宏,吕建,金芝.浅论软件技术发展.电子学报,2002,30(12A):1901−1906. [doi: 10.3321/j.issn:0372-2112.2002.z1.001]

[2] 杨芙清,吕建,梅宏.网构软件技术体系:一种以体系结构为中心的途径.中国科学(E 辑:信息科学),2008,38(6):818−828. [doi: 10.1007/s11432-008-0051-z]

[6] 陈伟,魏峻,黄涛.W$^4$H:一个面向软件部署的技术分析框架.软件学报,2012,23(7):1669−1687. http://www.jos.org.cn/1000-9825/4105.htm [doi: 10.3724/SP.J.1001.2012.04105]

[29] 张颖,黄罡,刘儇哲,等.一种支持 Java 应用中计算按需远程执行的方法.软件学报,2013,24(8):1713−1730. http://www.jos.org.cn/1000-9825/4344.htm [doi: 10.3724/SP.J.1001.2013.04344]

**蔡华谦**(1990－),男,福建石狮人,博士生,主要研究领域为移动计算,软件中间件.

**黄罡**(1975－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为软件中间件,软件体系结构,网构软件.

**张颖**(1983－),男,博士,讲师,CCF 专业会员,主要研究领域为软件工程,软件中间件,自治计算,分布式计算.

**梅宏**(1963－),男,博士,教授,博士生导师,中国科学院院士,CCF 会士,主要研究领域为软件工程,软件中间件,软件体系结构.