# FOUR-LAYER CAKE: SEPARATING ADVERTISEMENT FROM HOST APPLICATION ON ANDROID

BY

LINGYU XU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor Carl A. Gunter

# ABSTRACT

Many applications on Android rely on advertisements for revenue. In the current advertisement model, ad libraries are linked to host applications and their permission requirements are coupled. More permissions means more targeted ads, which brings more revenue. As a result, developers tend to seek more permissions from the user, which is not desirable with regard to the user's privacy. In this thesis work we attempt to address two approaches to solve this problem. The first separates ad library permission requirements from the host application, and the second provides users with a four-level privacy-concerned advertisement mechanism.

Though developing the ad module and the host app in two different applications makes sure that the permission requirements are separated, the ad module requires interaction with the user through the host app. The Android system does not yet support cross-application embedding and interaction. In this thesis work, we build our model based on an existing work called LayerCake, which supports secure embedded user interfaces by modifying the Android system, allowing the host application to embed another activity that runs in a separate process.

We propose to provide users with four levels of ads to choose. Highest Privacy Level (Level 3): No Ads, which means the user makes a payment directly to the provider to get rid of the ads; Fundamental Privacy Level (Level 2): Plain Ads, which means the advertiser broadcasts ads without targeting; Pragmatism Privacy Level (Level 1): Inter-app Ads, which means we utilize installed packages information to select advertisements; Trusted Privacy Level (Level 0): In-app Ads, which means we gather user information in the host app to provide more targeted ads.

Our work, Four-Layer Cake, using the above two approaches, effectively creates an architecture that Android users are aware of how their information are collected and used, so they can select their own privacy and service level.

*To my parents, my advisor and my friends, for their love and support.*

# ACKNOWLEDGMENTS

First and foremost, I'd like to express my appreciation to Professor Carl A. Gunter, my advisor, who has always been guiding me through these two years and has been offering tremendous helps to me, especially at the most difficult moments. I'm really grateful for the learning experiences in the University of Illinois at Urbana-Champaign and the opportunities to study from different people that my advisor has given me.

I also appreciate the help from, including but not only, Soteris Demtrious, Muhammud Naveed, Vincent Bindschaedler, Yunhui Long, Aston Zhang in the Illinois Security Lab, Dongjing He, Xun (Sean) Lu, who has graduated from the University of Illinois at Urbana-Champaign. They are among the most outstanding students in our department and are really helpful. When I got stuck in my study and turn to them, I can always get kind help. In particular, Yunhui and I did the course project for *Advanced Computer Security* together and I really cherish the opportunity to gain deeper insights into privacy issues on Android during our discussions and project meetings with our advisor, Professor Carl A. Gunter.

Last but not least, I would also like to express my love and thankfulness to my parents. Without their unconditional love and support, I would not have had the chance to study in the University of Illinois at Urbana-Champaign and gain so many valuable experiences.

# TABLE OF CONTENTS

LIST OF TABLES . . . vii

LIST OF FIGURES . . . viii

CHAPTER 1 INTRODUCTION . . . 1
- 1.1 Overview . . . 1
- 1.2 Problem Statement . . . 2
- 1.3 Our Approach . . . 2
- 1.4 Main Contributions . . . 3
- 1.5 Outline . . . 4

CHAPTER 2 BACKGROUND . . . 6
- 2.1 Current Design of Advertising Model and Its Limitations . . . 6
- 2.2 Known Attempts to Preserve Privacy on Android Platform . . . 8
- 2.3 Known Uses of App Bundles . . . 10

CHAPTER 3 ANDROID LAYERCAKE AS A PLATFORM . . . 12
- 3.1 LayerCake Features . . . 12
- 3.2 LayerCake Performance when Embedding Advertising . . . 14

CHAPTER 4 IDEAS RELATED TO APP BUNDLES . . . 16
- 4.1 Sequential Pattern Mining for Prediction in Medicine Field . . . 16
- 4.2 Amazon Anticipatory Package Shipping . . . 17
- 4.3 Persistent Cookies Profiling . . . 19

CHAPTER 5 FOUR-LAYER CAKE DESIGN . . . 21
- 5.1 Related Work . . . 21
- 5.2 Modified Three Parties Feedback Loop . . . 22
- 5.3 Four Layers of Privacy . . . 23

CHAPTER 6 IMPLEMENTATION . . . 25
- 6.1 Advertisement Library Server Implementation . . . 26
- 6.2 Android App Client Implementation . . . 27
- 6.3 Advertising Procedure . . . 31

CHAPTER 7 DEMO AND EVALUATION . . . . . . . . . . . . . . . . . 35
7.1 Demo . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
7.2 Evaluation and Analysis . . . . . . . . . . . . . . . . . . . . . 40

CHAPTER 8 CONCLUSION AND FUTURE WORK . . . . . . . . 42

REFERENCES . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

# LIST OF TABLES

7.1 Advertisements Used in the Demo . . . . . . . . . . . . . . . . . 35

# LIST OF FIGURES

2.1 The current advertising model . . . . . . . . . . . . . . . . . . 6

4.1 How does Amazon Anticipatory Package Shipping Work . . . 18

5.1 Three-Party Feedback Loop Model for Android Advertising . . 21
5.2 Modified Three-Party Advertisement Model for Android . . . 23
5.3 Four-Layer Cake Privacy Model for Android . . . . . . . . . . 24

6.1 Advertisement Library Server . . . . . . . . . . . . . . . . . . 26
6.2 Android Application Client . . . . . . . . . . . . . . . . . . . 28
6.3 Communication between Host App and Ad App process . . . 29
6.4 Communication process between Ad Library Server and Android Client . . . . . . . . . . . . . . . . . . . . . . . . . . 32
6.5 Advertising Process and State Transition Finite State Automaton . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

7.1 Ads Loop 1 : Health-related (AppType 0) Ads . . . . . . . . . 36
7.2 Ads Loop 2 : Music-related (AppType 1) Ads . . . . . . . . . 36
7.3 Click on Ads to Show Application in Google Play Store . . . . 38
7.4 Unlock Professional Functionality via Payment . . . . . . . . . 39
7.5 Unlock Professional Functionality via Payment . . . . . . . . . 39

# CHAPTER 1

# INTRODUCTION

With the prevalence of the current design of application and advertisement model, the resulting problems are attracting more and more attention. In the current advertisement model, ad libraries are linked to host applications, and they may require additional permissions to gather user personal information on the devices to provide more targeted ads. These permission requirements are then directly presented to the users as requirements of the host applications. The current advertising business model accepted in the Android market is the main mechanism that helps fund the exploding emerging of mobile phone applications. Regardless of the fact that this model has been prevalent in the Android application market, it has significant drawbacks with regard to the protection of user privacy. As more detailed user information tends to results in more targeted ads, more targeted ads brings about more revenue, the developer is likely to seek more permissions from the user, an ad library tend to abuse a host application's permissions. Another drawback of this current model is for the advertiser: a malicious application could simulate interactions between the advertisement and the user to cheat the advertiser in order to make more revenue.

## 1.1 Overview

In Android on-line advertising, there are a number of models which aim at privacy-preserving erupting in recent years. They are able to provide privacy protection via effective ways such as delivering mock information to the application [1, 2] or using differential privacy to avoid profiling due to details [3]. Most of these models have been focusing on the preserving of privacy and are successful to some extent, but have given too little attention to the needs of a huge party in the advertising market: the advertisers. The fact that the implications for a market is largely driven by the accurate profiling of users is merely considered [4]. These models are either unable to provide accurate ad profiling or are too complicated that advertisers find it difficult to adopt. As a result, there is hardly any practical model existing that could properly balance between Android user privacy protection and advertisement profiling. This project is trying to solve this problem by applying the idea of privacy-level negotiation into current privacy-protection advertising models.

## 1.2 Problem Statement

The market of Android on-line advertising contains three primary parties: advertisers, who make profit by selecting targeted advertisements using accurate user profiling information; developers, who receive reward/revenue from advertisers by providing accurate user information for profiling; and users, who provide their private information in exchange for services offered by developers. A practical privacy-preserving advertising model should equally consider the interest of all three parties in the market. Balance is so difficult to achieve which calls for several specific questions:

1. ***How is the negotiation between advertisers, developers and users expected to take place?***
2. ***How should we specify different levels of privacy and relate them with different service levels?***
3. ***What are the incentives for advertisers, users and developers to use the negotiation model?***

The solve of these questions will lead to the balance the benefits of all three parties and achieve our primary goal of privacy-preserving.

## 1.3 Our Approach

To answer the first question, in this work, we propose to take advantage of LayerCake [5]—a modified version of the Android System [6]—to clearly separate Host application and Advertisement Module permission requirements.

By doing this the developer has no motivation to require unnecessary permissions from the user. The separation of the permission requirements would provide the users with a clear and clean overview of what information is being collected and how and where is the information being used. Information will flow to the developer and the advertiser in two individual flows.

For the second question, we introduce a new advertising model from which each of the three parties benefits. To induce the users, we illustrate that the advertiser get information from the user directly with out revealing any unnecessary private information to the provider and user can choose different privacy levels by themselves. To encourage the developers to accept the new model we validate that when they provide more service they get more revenue by establishing a indirect link between service and revenue. On the advertiser side, as they get needed information directly from the user, they could circumvent being cheated by any application developer.

In the meanwhile, with respect to the third question, we explore methods to deliver ads due to *Inter-app* and *In-app* information, which specifies 4 clear levels of privacy. Inter-app information—the list of installed application packages, including application name, package name, version name, version code, application icon, application installation (or modification) time, etc—is also referred to as *App Bundles* in the rest parts of this thesis work. We are also employing In-app information—demographic information and in application behaviors collected by the host application—in our advertising selection algorithm.

## 1.4 Main Contributions

Some of the major contributions of our work are listed as follows:

- ***Three Party Loop Module:*** First of all, the permissions required by the host application and the advertising application are aimed to be separated. Users only provide the necessary permissions to the host app, and all the additional permissions should be independently required by the advertisement module. Advertising app delivers the users' privacy information ( demographic information like age and gender, in app behavior information, long term track of click through rates, etc) to the ad network server, then the ad server pushes selected advertisement to be displayed to the users, and give corresponding revenue to the app developers. Taking all these requirements into consideration, we make modifications to the model proposed in [4]. Service from the application provider, information flow from the user, and revenue from the ad network, forms a three-party feedback loop. The ad network is also responsible for delivering advertisements to the embedded advertisements application.

- ***Exploit of App Bundles:*** To our knowledge, there are not many advertising system that exploit the utilization of App Bundles. Twitter is collecting the installed packages on Android devices [7], Facebook is also utilizing similar information [8] but only gathered from applications that have their code baked into them. We note the importance and significance of App Bundles, and when comparing to buying products from a super market, the usage of App Bundles to predict the next possible application installed is similar to sequential pattern mining in Data Mining field [9, 10]. We use App Bundles as the as a main factor of selecting advertisements on the ad server side, and we use In-app information for further filtering the ads.

- ***Offering four levels of privacy associated advertisement:*** Depending on the user's preference, we provide four privacy concerned service and advertisement related levels. With regard to privacy revealing extend, the four levels include: professional functionality with no ads, basic functionality with plain advertisements (broadcast from the ad server without any targeting), medium functionality with Inter-App targeted Ads by collecting App Bundles information, professional functionality with In-App targeted Ads based on personal information collected by the host application.

## 1.5 Outline

We give an outline of the following contents of this thesis.

Chapter 2 gives the thorough background information on the Android online advertisement market. In particular, section 2.1 introduces the current design of the advertising model and argues its limitations; section 2.2 describes some known attempts of providing approaches of splitting ads from the host apps to preserve user privacy; section 2.3 brings in the idea of App Bundles and introduces some know attempts of utilizing App Bundles.

Chapter 3 aims to understand a modified version of Android System, LayerCake, since we build our model based on this platform. Section 3.1 analyzes the primary features of the LayerCake system, while section 3.2 introduces how LayerCake benefits Android advertising.

Chapter 4 illustrates the concept of App Bundles and discusses why and how could it be utilized. Sections 4.2, 4.3 respectively discuss the similarity between App Bundles and Amazon Anticipatory Package Shipping and Persistent Cookies Profiling.

Chapter 5 explains the design of our model, Four-Layer Cake. Chapter 6 gives the implementation details of our mechanism while chapter 7 offers the evaluation demo and analysis. Last but not least, Chapter 8 concludes our work, analyzes the limitations and proposes possible future works.

# CHAPTER 2

# BACKGROUND

This chapter gives background information of the current advertising model in Android market and the existing attempts of preserving user privacy in Android on-line advertising. In particular, section 2.1 explains the current design of advertising model, section 2.2 describes known exploits of preserving user privacy on Android platform, and section 2.3 gives examples of how App Bundles information has been used.

## 2.1 Current Design of Advertising Model and Its Limitations

[4] investigate the current advertising model by analyzing a popular ad-network Admob [11]. In its generic form, this advertising model involves three primary parties. First, the user, who receives service provided by the mobile application developer. Second, the developer, who expects to get compensation/revenue from the advertiser as a reward for the delivering its service. Third, ad-network, who pays the developer when it successfully gathers the user's preferences and delivers targeted adverts.

Figure 2.1 shows the relationship of the three parties in the current model.

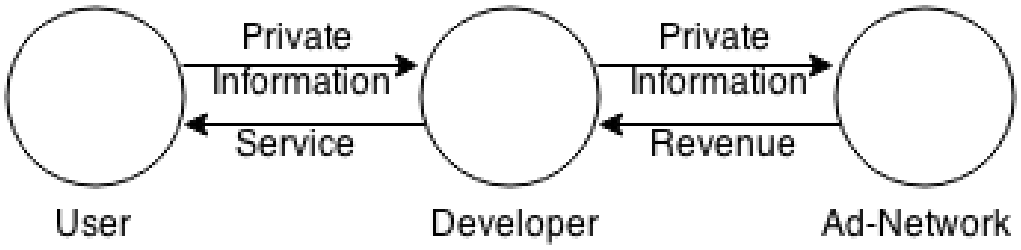


Figure 2.1: The current advertising model

In this model, one thing worth noting is that all the private information of the user first flows to the developer, regardless of what does the developer actually needs. The complete delegation of information collection means it is possible that the user is unwilling to expose some of information to the developer.

In Admob ad-network, the developer is paid according to an advert's "click-through rates". An impression of the advert displaying in the Android application may lead to a "click" from the user. A "click" means the user is attracted by the ad and proceed to the download stage. The success rate of the delivered ads means the successful operation of the advertiser, and will significantly leads to revenue rewarded to the developer. Consequently, Admob ad-network as well as the provider aim to generate as many "clicks" per "impression" as possible. When the ratio of "clicks/impression" is high, it means that the adverts are well targeted at the user. And since the advertiser need to pay the developer anyway for making the impressions to the user, the cost of unsuccessful advertising is reduced when the ratio is high. Demographic information (such as gender, age), location information and social networks information are some of the kinds of information used in user profiling algorithms for advertisement targeting.

More targeted ads means more profit and less cost, so an advertiser may be incentivized to collect as much information as possible about the users for accurate profiling. For the developer, it will only be rewarded when the impression of an advert gets a "click". The tight coupling of the advertisement module and the host application shown in Figure 2.1 results in a fact that the developer tends to support the profiling algorithm on the advertiser's side. This model actually could not meet the trust expectation by the user, and lead to the consequence that the privacy requirements of the applications rarely represents the true demand of the service provided. There should be a pressure on the developer to constraint its request of unnecessary permissions, or more effectively, a way of keep it away from touching the user's privacy.

## 2.2 Known Attempts to Preserve Privacy on Android Platform

There are some existing work providing some level of privacy protection to the user, here we mainly introduce two categories:

1. Delivering mock information to the host application.
2. Splitting the advertising module from the host application.

MockDroid [1], a modified version of the Android system, is a representative of category 1. It allows a user to choose to preserve his personal information. In MockDroid, a user could 'mock' the access to a resource just like an application. After this 'mock' access, the resource accessed would be marked as unavailable subsequently. Whenever an real application requests to access the resource, it gets a report that the particular resource is empty or unavailable. This mechanism helps the user to choose what kind of personal information that could be exposed. In the meantime, the user can be encouraged to think about the trade-offs between the service he gets and the disclosure of private information, as the application that request the access to a certain resource could have provided better service if the access is authorized. Though this kind of approach provides privacy preserving to some extent, the simple forbidding of resource access would not work well the the Android on-line advertising market, since it has not take the interest of the advertiser into account. The ignorance of the advertiser's profit will cause the whole advertising market to collapse.

The balance of interest of the three parties in the advertising model is fundamental. To this end, we look at the second category: attempts of splitting advertisement module from the host application. The deployment of methods in this category still ensures that the advertiser gets profit.

[4] proposes a framework which includes a market mechanism and establishes a three-party virtuous feedback loop. This feedback loop works well to balance between the privacy information collected from the user and the service provided to the user. In a word, the user exchanges more privacy information for more services. For the provider, the more service it provides it gets more rewards. For the advertiser, the more profiling information collected, the more accurate the advertisement targeting would be. None of the

three parties can abuse its ability of controlling the resource that it owns. In this model, the private information needed by the advertiser directly flows to the ad-network, instead of being collected by the developers first. The ad-network therefore gives the corresponding revenue to the developer according to the click through rates of the advert. The developer provides different levels of services due to the resulting revenue. A redesigned application market is also playing a important role. The new market will ask the users to rate applications, those applications who ask for unreasonable permissions will get low scores, and the most unwanted permissions will be listed as most conspicuous. With the help of peer-pressure from a redesigned market, the developers can be incentivized to not collect unnecessary privacy information from the users. The proposed framework is implemented as a separate advertisement service which allows other applications to subscribe to. A real time monitor is responsible for monitoring the data flows between the three parties, recording the overall clicks and controlling revenue. There are, though, limitations of this mechanism. This design does not allow the user to choose what kind of information can be exposed. The upgrade of service level and the privacy exposure level may not be out of the user's willingness.

Adsplit [12] proposes an approach to automatically separate an application and its advertisement module, thus allows them to run in different and independent processes under different user-ids. In this way, permissions required by advertisements can be separated from those required by host applications, giving the user a clear prospect on how their privacy information is used. This function of AdSplit is achieved by providing AdWebview, a built-in advertisement application in Android core distribution. AdWebView could load HTML and Javascript from advertising libraries, fetch advertisements accordingly, and display the advertisements on a WebView component in separated process and activity. It supports three categories of permissions: the permission to load a url, the permission to call to HTML5 geo-location API and the permission to maintain long-term tracking cookies. However, the process of granting permission is non-negotiable. Users would either grant all the permissions required by an advertiser or have all the permissions denied. Also, it is unclear how host applications could respond when advertisement permissions are denied. Adsplit successfully prevents programmatic click-fraud attacks using Quire [13], by authenticating user input.

LayerCake [5] explores the requirements to support secure embedded user interfaces of a system. The researchers analyze existing systems such as browsers and smart phones systematically, with regard to whether they provide security properties and how do they provide these properties if any. They make a modification to Android system (Android 4.2 JELLY_BEAN_MR1) and end up with being capable of supporting secure interface embedding. They implement their design and evaluate the implementation using case studies which are based on embedded interfaces. Advertisement libraries and Facebook social plugins, which are typical types of embedded widgets, are evaluated. The LayerCake system has been deployed in some of the recent works of the research group from University of Washington, showing in [14] and [15].

## 2.3 Known Uses of App Bundles

The news of Twitter [7] starting to track user's installed packages for advertising targeting purposes quickly exploded on November 26th, 2014 [16, 17, 18]. The app tracking mentioned in the above news, means that Twitter application on Android, is tracking lists of applications installed and the in-app basic metadata to use for advertising. As is mentioned in 1.2, the list of applications installed, we will use *App Bundles* to refer to it in the rest of this work.

The fact that the related API [19] is built in the Android Operating System makes it extremely easy to get the App Bundles. What's more, the calling of this API does not even require any special permissions. A simple call of *getPackageManager().getInstalledPackages()* returns the developer with all the information about installed packages on a device, including application name, package name, version name, version code, application icon, application installation time, etc. There are even more ways provided by the Android API to refine the results, for example, a call to *getPackagesHoldingPermissions()* returns a list of all installed packages that are currently holding any of the given permissions on the device. Package manager can also pull information about a particular individual application once it is determined this application has already been installed. Twitter claims that they have only been collecting and updating data of applications that the users have installed, but not using any in-app information [7]. In our work we propose to use in-app information from our host application to further tailor the result set of advertisements.

To our knowledge Twitter is a pioneer of social networks who collects such information on users devices, but maybe not the only one. Facebook has been reported to being collecting similar data [8], but only from applications that have their code baked into them. This is referred to as “software development kit” [20].

# CHAPTER 3

# ANDROID LAYERCAKE AS A PLATFORM

Smart phone applications nowadays commonly embed third-party user interfaces such as advertisements, Facebook social plugins, and access control gadgets[5]. The capability of embedding third party user interfaces comes with security issues, and Android does not yet support secure cross-application interface embedding. The LayerCake [5] group proposes to modify the Android operating system to provide support for secure embedded user interfaces from scratch. Our work, Four-layer Cake, uses LayerCake as a platform to build our models on, so this chapter gives an introduction of what main features does LayerCake provide and how does it work when embedding advertisement module.

## 3.1 LayerCake Features

LayerCake, a modified version of the Android system, supports secure cross-application embedding via making changes to the ActivityManager class, the WindowManager class, and input dispatching. Some features of LayerCake that are fundamental to Android on-line advertising will be listed as follows:

- Allow more than one application to be visible to the user at the same time.
- Allow one application to embed one or multiple instances of other Activities which does not interfere with each other.
- Provide clickjacking prevention and ancestor redirection prevention.

### 3.1.1 Visualization of More Than One Application

Android UI provides the user with a particular view for an application. An application may consist multiple Activities, each of them defines built in View elements. The original Android operating system's ActivityManager could only keep one Activity in the foreground. An application is not capable of embedding an Activity from a different application. Though Android ActivityGroups provide UI code reuse within the same application, it does not support true cross-application embedding. LayerCake explores to allow one application to embed Activities from a different and independent application. The embedded Activities run in a separate process. Android's WindowsManager isolates the window of each application, i.e., an application cannot access the window from another application, nor could it dispatch the user input for the other application. The isolation properties are relied on by the LayerCake design. In LayerCake, multiple applications may have visible windows, though only one application could be in the foreground, and their interaction with the user are clearly isolated.

This allows the advertisement module in Four-Layer Cake to share the view with the host application, in a secure cross-application embedding way.

### 3.1.2 One Host Embedding Multiple Applications

A new View called ***EmbeddedActivityView*** is introduced into Android's UI toolkit. This View allows the embedding of another application's Activity to a host application. Specifying the package and class names of the embedded Activity in the parameters of the EmbeddedActivityView ensures the success of embedding in the host application's interface.

Android's ActivityManager is extended to support embedded Activities. These embedded Activities are launched when the corresponding instances of EmbeddedActivityView is created and displayed. One thing is that the embedded Activities are not under the control of ActivityManager, they follow the life cycle of the embedding Activity.

Multiple Activities may be embedded in the same host application. The embedded Activities themselves may also embed one or more Activities. What's more, multiple instances of the same Activity may be sharing the same host. An application may also be embedded by multiple host applications, thus allowing the leverage of information collected inside different hosts.

The fact multiple embedded activities are supported makes it possible for an Android application to embed advertising module for different advertisement networks.

### 3.1.3 Clickjacking Prevention

A clickjacking attack [21] happens when a malicious application tricks or compels a user into interacting with an interface. For example, a malicious application may make an interactive UI element as transparent, thus the user has the chance of passing through inputs into it with out knowing anything. LayerCake is able to prevent clickjacking by checking the following:

1. Whether the Activity is covered by another window (obscured).
2. Whether the minimum requested size is not met.
3. Whether the Activity is not fully visible due to window placement (a View could be cropped due to scrolling).

If any of the answers to the above question is “YES”, LayerCake would just discard the embedded Activity’s user input.

These rules help prevent the application providers from cheating users for more clicks.

## 3.2 LayerCake Performance when Embedding Advertising

In the current Android design, stock Android applications embeds third-party advertisements and provides an AdView element to show the adverts, and ad library module is running in the same process as the host applications.

In LayerCake, modifications have been made to separate the AdView out into an individual process. In the evaluation demo, a wrapper application for AdMob [11], a primary advertisement library, is created as an embeddable application. All of the APIs of the ad library is exposed across the process boundary, making it possible for the host application to pass on parameters to it.

LayerCake is successful in moving ads into an individual process and addresses a number of concerns. First and most importantly, permissions needed by the advertisement library would no longer be requested by the parent application. Second, an ad library could also no longer abuse a host application's permissions [22]. Thirdlym the fact that all ads from the same ad library—even if the Admob application is embedded in different host applications—run in the same process allows the Admob wrapper application to leverage input from different host application sources. Last but not least, the host application can no longer be capable of mounting programmatic click fraud attacks.

# CHAPTER 4

# IDEAS RELATED TO APP BUNDLES

Twitter is most likely to be the pioneer of exploiting the usage of App Bundles. There are, though, a small number of know attempts of using this information.

App brain[23] is an online Android Market which allows users to upload lists of installed applications to it for management. App Usage Tracker [24], an application available in the Google Play Store [25], tracks how all the installed apps are getting used on the device, and depicts the usage information in a graphical format. We can come to the conclusion that the usage of App Bundle information in Android on-line Advertising is still in its early stage, and we need to further explore the features of it.

Though the usage of App Bundles still needs exploration, there are examples in other areas that provides similar attempts of using known information to make predictions.

Three concrete examples are given as follows: section 4.1 introduces that the idea of sequential pattern mining has been used for prediction in the medicine field, section 4.2 illustrates how does Amazon anticipatory package shipping work, section 4.3 finds the similarity between App Bundles and persistent cookies.

## 4.1 Sequential Pattern Mining for Prediction in Medicine Field

Sequential pattern mining [9] is a data mining technique which can be used to identify patterns of sequenced events within a database. The original application of sequential pattern mining is in the retail industry: after purchasing a particular book, a customer is predicted to buy its sequel within a certain time period.

Applications in medicine field were proposed in [26], and have received great success in disease susceptibility prediction [27]. [28] uses sequential pattern mining to establish temporary links between medications automatically. The links are visualized and used for generating rules to predict the next possible medicine prescribed to the patient. [28] does evaluations respectively at drug level and drug class level, and come to a conclusion that frequent pattern mining is effective in identifying the temporary links between medicines and predicting the next prescribed medicine.

As is mentioned in Section 1.2, App Bundles information includes application name and application installation (or modification) time, in our work we also classify applications into different types. When being compared to the prediction in medication prescription, an application is a "medicine", its installation time is the "prescribed time", and its type information is the "drug class" in medication.

## 4.2 Amazon Anticipatory Package Shipping

A news titled *Amazon knows what you want before you buy it* [29] claims that Amazon could conceivably use a patent for the algorithm-based system to ship products even before the customer place an order.

Officially known as "method and system for anticipatory package shipping" [30], the benefits of the system are obvious: the accurate predicting of customers orders helps increase sales. Further more, the potential money and time cost of shipping could be largely reduced.

Figure 4.1 shows how the Amazon anticipatory package shipping work.

According to the patent, the prediction model is using data from a user's previous Amazon behaviors, including but not only, time on-line, links clicked in site, duration of views, wishing list, shopping history and shopping cart status. The algorithm also takes real-world customer personal information into account. These information can be collected from customer telephone inquiries, responses to advertising materials, and so on.

App Bundles could be made analogy to Amazon's anticipatory package

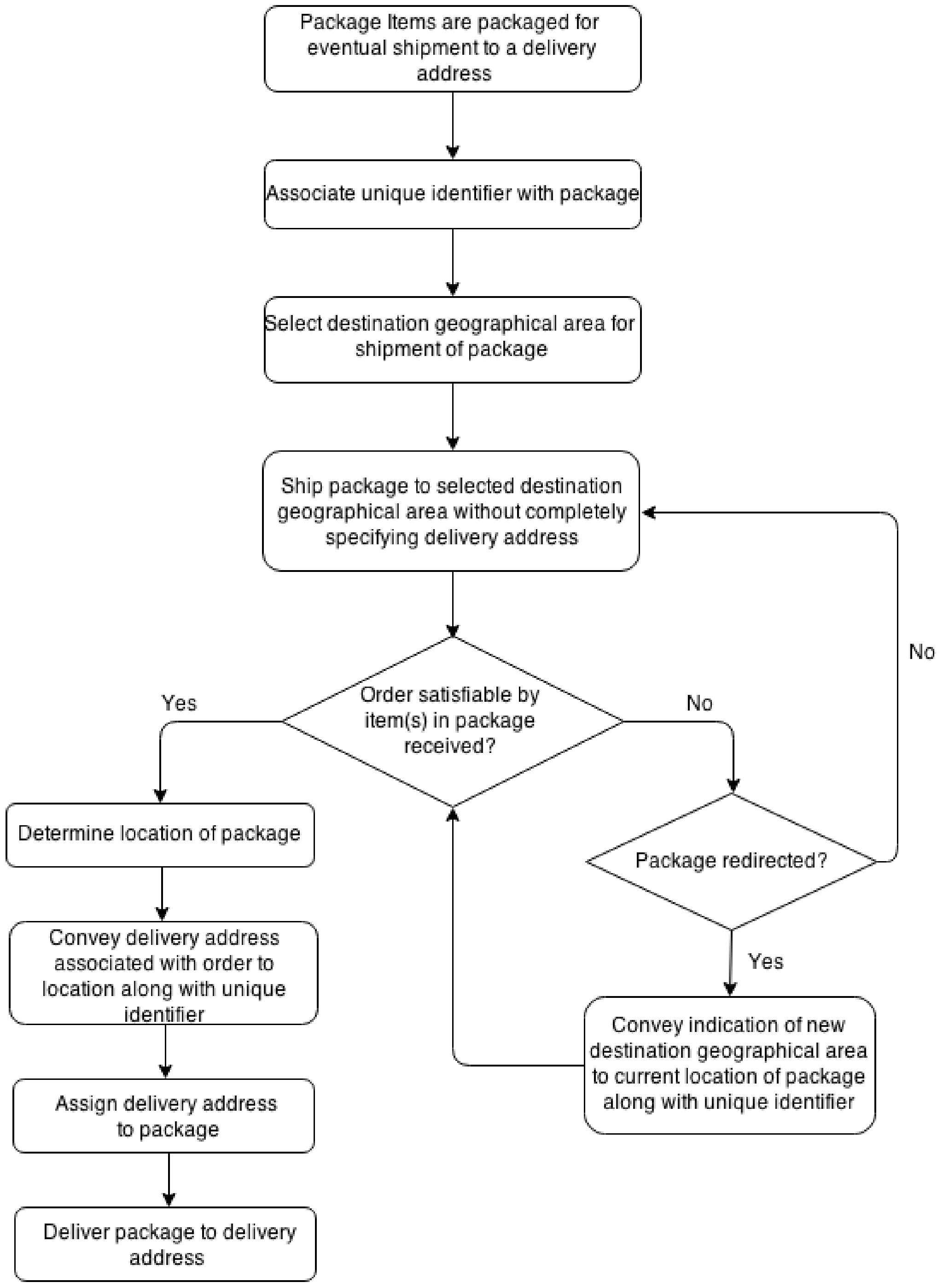

Package Items are packaged for eventual shipment to a delivery address
Associate unique identifier with package
Select destination geographical area for shipment of package
Ship package to selected destination geographical area without completely specifying delivery address
Order satisfiable by item(s) in package received?
Yes
No
Package redirected?
No
Yes
Determine location of package
Convey delivery address associated with order to location along with unique identifier
Assign delivery address to package
Deliver package to delivery address
Convey indication of new destination geographical area to current location of package along with unique identifier

Figure 4.1: How does Amazon Anticipatory Package Shipping Work

shipping patent to some extent. In our case, an application not yet installed is our “commodity” for sale, the App Bundle information is like the shopping history of Amazon. Amazon make predictions, or anticipatory package shipping officially, (partially) due to the shopping cart activities and we make our recommendations due to the “bought” applications. To this end, we could also take an insight into Amazon’s efforts and employ their algorithms in our future work, even though in chapter 6 we are employing a straight forward algorithm for demo the advertising procedure: select advertisements based on the installed packages and recommend similar applications.

## 4.3 Persistent Cookies Profiling

Cookies has been working as a tagging mechanism of identifying a user out of millions. Cookie profiling uses persistent cookies to track the overall activities of a user online.

Cookies tracking is occurring whenever you are browsing [31] web pages. Marketers/advertisers are most likely to do cookies profiling. They collect and collate information about a certain user from cookies and create the “profile” for him. The behaviour of a user when browsing the Internet becomes the reason why he has been targeted by a particular collection of adverts [32].

Cookie profiling is the only way for marketers/advertisers to target potential customers and obtain a possible product purchase from them. By knowing a users browsing habits, including sites visited, age, gender, marital status, political preferences and religious affiliations, they can show him or her advertisements that are appealing, advertisements that he or she will care to patronize. This is a way for marketers/advertisers to increase their profit and cut down on the cost of unsuccessful delivery of adverts by accurately targeting their customer.

The advantage of using cookies to do the on-line profiling is that it is permitted by users to some extent, since the profiling using cookies is less alarming and less offensive than, for example, buying data from social networks.

There could also be a line drawing from Four-Layer Cake App Bundles usage to Cookies profiling as analogies. Both of them are used for on-line advertising purposes, though one in web browsers and the other in Android applications. App Bundles is just like the persistent cookies, which stores the past “behavior” of the user, and with a careful designing of algorithm, can make accurate predictions about a user. The App Bundles is also a long term information as an Android device is always having some installed applications. In our approach we are also using in-app information gathered by host application to make more targeted ads. An obvious thing is that, the more in-app information we get, the more accurate the adverting would be. This is actually similar as the multi-source cookies used for profiling in the web browser advertising.

# CHAPTER 5

# FOUR-LAYER CAKE DESIGN

Our work mainly includes two core parts: the three parties feedback loop and the four levels of privacy. Section 5.1 illustrates the related work for our design, section 5.2 introduces the idea of a modified three party feedback loop, section 5.3 gives the idea of four privacy levels with regard to user preferences.

## 5.1 Related Work

As is mentioned by figure 2.1 in section 2.1, in a typical in-application advertisement under the current advertisement model, the collecting of user demographic information is delegated to the host application.

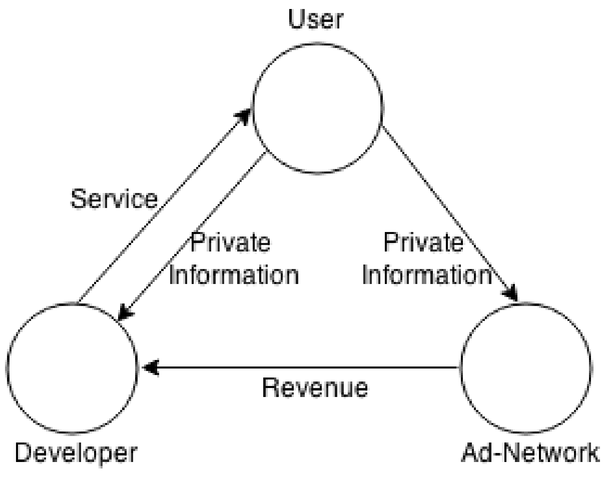


Figure 5.1: Three-Party Feedback Loop Model for Android Advertising

The tight coupling of the ad-network module with the host application brings the result that the permissions required for advertisement targeting are publicized as part of the host application's permission requirements. The obscuring of who might access the users' personal information and how these information would be used causes privacy concerns.

To resolve this problem, [4] proposes a decoupled three-party feedback loop model, shown in Figure 5.1. In this new model, decoupling privacy control between the advertisement component and the host application is achieved. There are two independent information flows flowing respectively towards the ad-network and the host-application. This separation allows users to have different sharing agreements with the other two parties, and makes it easier for users to be aware of how is their information used.

[4] achieves the decoupling of application and advertising permissions by separating the two functions into distinct binaries. A generic advertising service which requests its own set of permissions is implemented. This service exports a new Intent for other applications to subscribe to.

Our work keeps the idea of three party feedback loop but does some modifications to employ LayerCake [5] to do the permission separation. LayerCake modified the Android system source code to support secure cross-application embedding, more details have been illustrated in chapter 3.

## 5.2 Modified Three Parties Feedback Loop

To balance user privacy and still protect the benifits of developers to ensure that they get deserved rewards for delivering the service, we establish a clear three-party model and form a feedback loop between each pair of parties.

Our idea can be illustrated by Figure 5.2.

First of all, the permissions required by the host app and the advertisement app, which is used to display the advertisements, are separated. Users only provide the necessary permissions to the host app, and all the additional permissions should be independently required by the advertisement application, which is running in a separate process. Advertisement application gets the users' privacy information and delivers selected advertisement to be displayed in the host application's *EmbededView* as a Banner Ad, and gives corresponding revenue to the host app developer. We adopt LayerCake to support secure embedded interfaces such that the host application and the ad application can be developed in two completely independent process.

We then propose to form a three-party feedback Loop to balance privacy

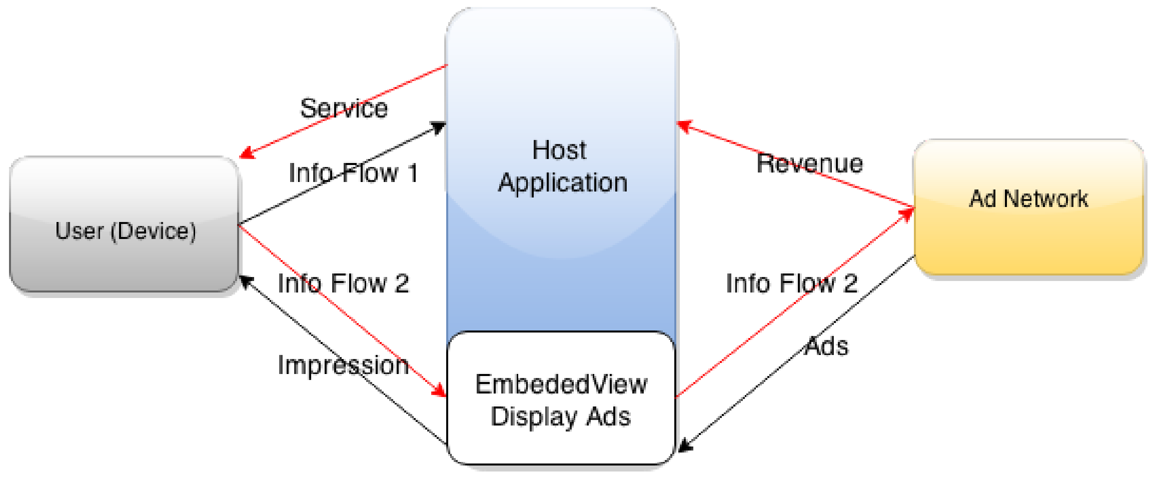


Figure 5.2: Modified Three-Party Advertisement Model for Android

and service (from the user's point of view). After the separating of permissions for the host application and the advertisements, we re-develop the applications and utilize LayerCake's feature to embed the ad application into the host. The information flows, service and revenue form a feed back loop between the user, the ad network and the developer (the loop is marked as red in Figure 5.2). The user can "pay" with privacy information in exchange for services, and the developer gets paid by encouraging the user to give more accurate privacy information. The advertisement application reports individual clicks to the ad network to calculate revenue to the developer. In this feedback loop, the advertisement application starts with plain ads, and may require more information from the user if the user requires more service (without paying off to the developer directly).

## 5.3 Four Layers of Privacy

We propose to provide users with four levels of ads to choose.

- Highest Privacy Level (Level 3): *No Ads*, when the user chooses to make a payment directly to the provider to get rid of the advertisements.
- Fundamental Privacy Level (Level 2): *Plain Ads*, which means the advertiser broadcasts ads without targeting.
- Pragmatism Privacy Level (Level 1): *Inter-app Ads*, which means App

Bundles information is used to select advertisements.

- Unconcerned Privacy Level (Level 0): *In-app Ads*, which means user information in the host app is gathered to refine ads to be more targeted.

Figure 5.3 shows the details of the four layers.

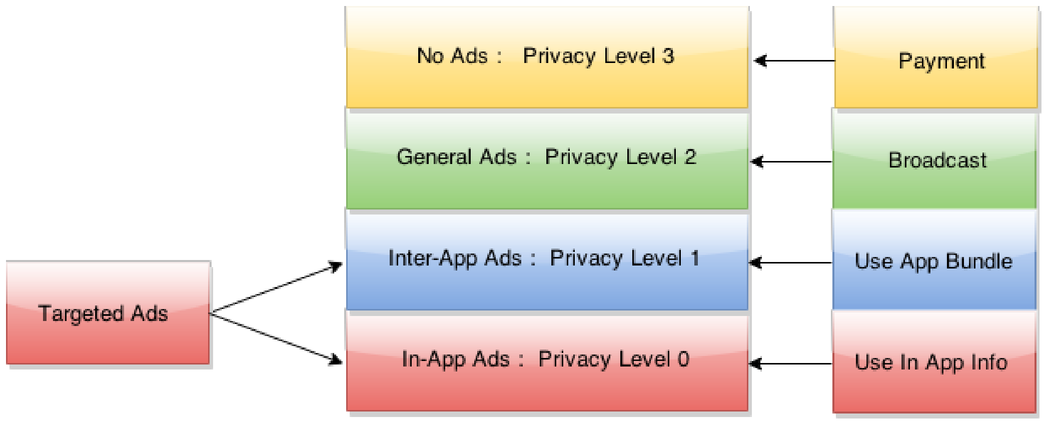

No Ads : Privacy Level 3
Payment
General Ads : Privacy Level 2
Broadcast
Targeted Ads
Inter-App Ads : Privacy Level 1
Use App Bundle
In-App Ads : Privacy Level 0
Use In App Info

Figure 5.3: Four-Layer Cake Privacy Model for Android

The feedback loop mentioned in section 5.2 starts with Fundamental Privacy Level (Level 2). No special information about the user or his device is provided to the developer or the advertiser, only plain ads are broadcast from the advertisement library server to each of the advertisement client connected to this server. If a user allows the usage of inter-app information for advertising purposes, the advertisement application will run for some time and collect all the App Bundles information which will then be reported to the server, and the user will be at Pragmatism Privacy Level (Level 1). Then if the user prefers to upgrade his service, he has two choices. To get rid of the advertisement and reach Highest Privacy Level (Level 3), he could make a direct payment to the service's provider; or, he can also choose to move to Unconcerned Privacy Level (Level 0) to get more service [33, 34] and more targeted ads. More details are demonstrated in chapter 6.

# CHAPTER 6

# IMPLEMENTATION

After understanding the mechanism and algorithms of our approach, this chapter gives implementation details of the Four-Layer Cake model. While the above model intuitively appears to be reasonable, it is not guaranteed that it provides satisfiable performance in practice. We explore what it takes in practice by demonstrating our model in chapter 7.

To provide a direct overview of the model introduced in this work, an advertisement library server and an application client on Android mobile phone are needed. We are not using any existing ad library, but building our own advertisement library server, to maintain more flexible query interfaces and provide modifiable advertisement recommendation strategy. Up to 5 threads are supported in our server.

On the Android client side, we are using LG NEXUS as a testing environment device. Android Layercake System is flashed into this device to support interactions and permissions separation between host application and its embedded activity.

Communications between the advertisement server and the android application are established via Socket. In section 6.1 we describe the Multi-thread advertisement library server implementation while section 6.2 describes the techniques we used on the Android Client side. Section 6.3 illustrates how the states of the advertising process change by presenting our socket communication finite state automaton. Since in our Four-Layer Cake model, both the Plain Ads Level and the Inter-app Ads Level require no additional permissions, for simplification, with regard to privacy Level, we just start from the No Ads (no privacy information provided to Advertiser), then Inter-App Ads (only use App Bundles information and requires no additional permissions), and then In-app Ads (requires personalized meta data collected by the host application).

## 6.1 Advertisement Library Server Implementation

Four-Layer Cake Model requires the support from a manageable advertisement library server. In our implementation, we used Java SE 1.6 development environment.

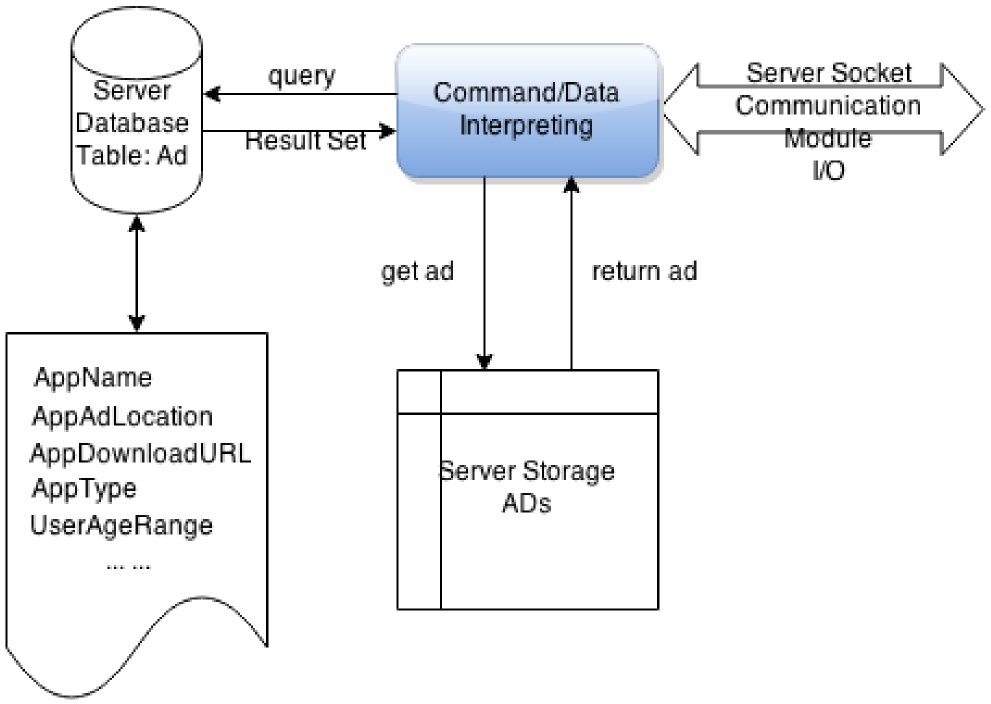


Figure 6.1: Advertisement Library Server

The server is expected to provide mainly four functionalities:

- Sending and accepting data flows to and from the client side.
- Interpreting data from bytes into strings or from strings to bytes.
- Maintaining database for advertisements information querying and retrieval.
- Storing advertisement files.

The coordination of the four modules are shown in figure 6.1.

First of all, the Communication Module. All of our commands and data are transmitted using Java Socket. In our code, the package /Server/src/socket includes SocketServer.java and SocketThread.java. SocketServer class helps maintain a thread pool of size 5, which allows up to 5 clients to connect to the advertisement library server. SocketServer is also responsible for accepting in-coming connections from clients and starting a new instance of SocketThread for each new client. SocketThread class provides individual and non-interfering threads for each client. It first waits until getting a connection request from a client, then it creates a new connection to the local database for afterwards queries. When it gets a command, since Socket communication transmits bytes, we need to interpret the command into string, using the embedded Command/Data Interpreting module. The advertisement results are also interpreted back into bytes and then sent to the client. The Server and the Client need to strictly follow the loop of sequences of *Command number–Data length–Data–Command number–...* when communicating, since Socket accepts all the data bytes as a stream, it cannot recognize individual commands or messages by itself.

Another package in our source code, /Server/src/operation contains DBOperation.java which provides interfaces of querying the database and returning result sets. The table *Ad* has the following keys: id, AppName, AppAdLocation, AppDownloadURL, AppType, UserAgeRange.

As Socket only transmits bytes, /Server/src/operation/Transfer.java provides the functionality of translating between strings and bytes. We are using ISO-8859-1 as the encoding standard.

The ads are stored in the server's local storage. The stored location of the ads in the result set of the database query is used to get ads from the local disk. The ads are returned to the interpreting module as picture file instances, and will then be interpreted into bytes and transmitted to the client via the communication module.

## 6.2 Android App Client Implementation

The Android application client implementation is based on the LayerCake System [5]. As mentioned before, LayerCake is a modified version of Android operating system which allows the using of secure embedded user interfaces and supports the host application to embed another activity that runs in a individual process.

The Android client includes two core parts. The host application provides functionality while the embedded ad application is responsible for displaying the ads and communicating with the ad library server. The information flows collected by the two applications are exactly separated since they are running in two independent processes. The relationship of the two parts are shown in Figure 6.2.

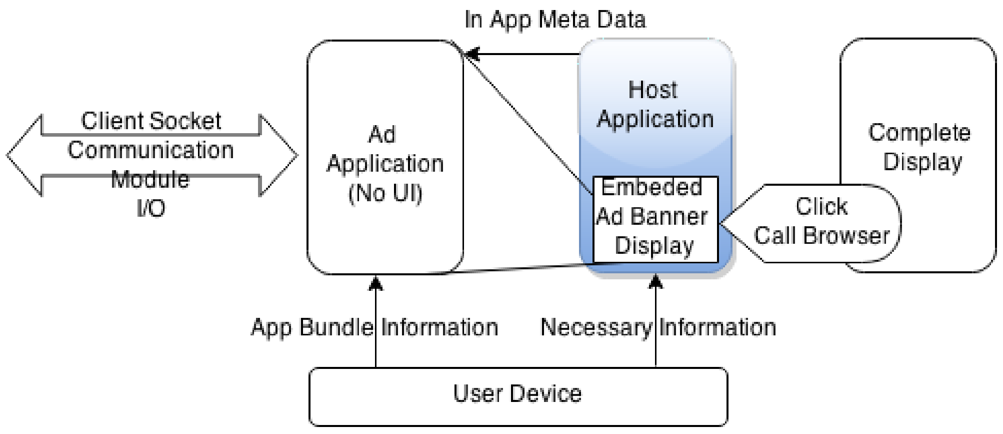


Figure 6.2: Android Application Client

The Host Application, in our test, is a health coach master which allows the user to keep track of the calories that he takes from meals, and the calories that he consumes when he works out. It only asks for necessary permissions upon installation and does not require additional permissions for the advertising module. The advertisement application is installed independently. When the user gets the basic version of the application, he is only offered an overview of his calories information and the UI to record his meals. Only after unlocking the exercise functionality could the user access the third tab to record how many calories he consumes via working out. There are two buttons in the Host Application provided to the user to have the third tab unlocked: making a payment (and also the user gets rid of the ads, so that the provider's revenue comes directly from the user's payment), or providing in-app personal meta data (and also get more targeted advertisements). The Host Application provides an *EmbeddedView* for the banner advertisement (which is enabled by the modified Android System LayerCake) to be displayed.

The ad application has a transparent UI of itself and is only visible to the user when showing in the host application. When it is embedded in the host application, the ad picture is retrieved from the ad library server and then displayed as a banner. When the banner is clicked on, a message prompts

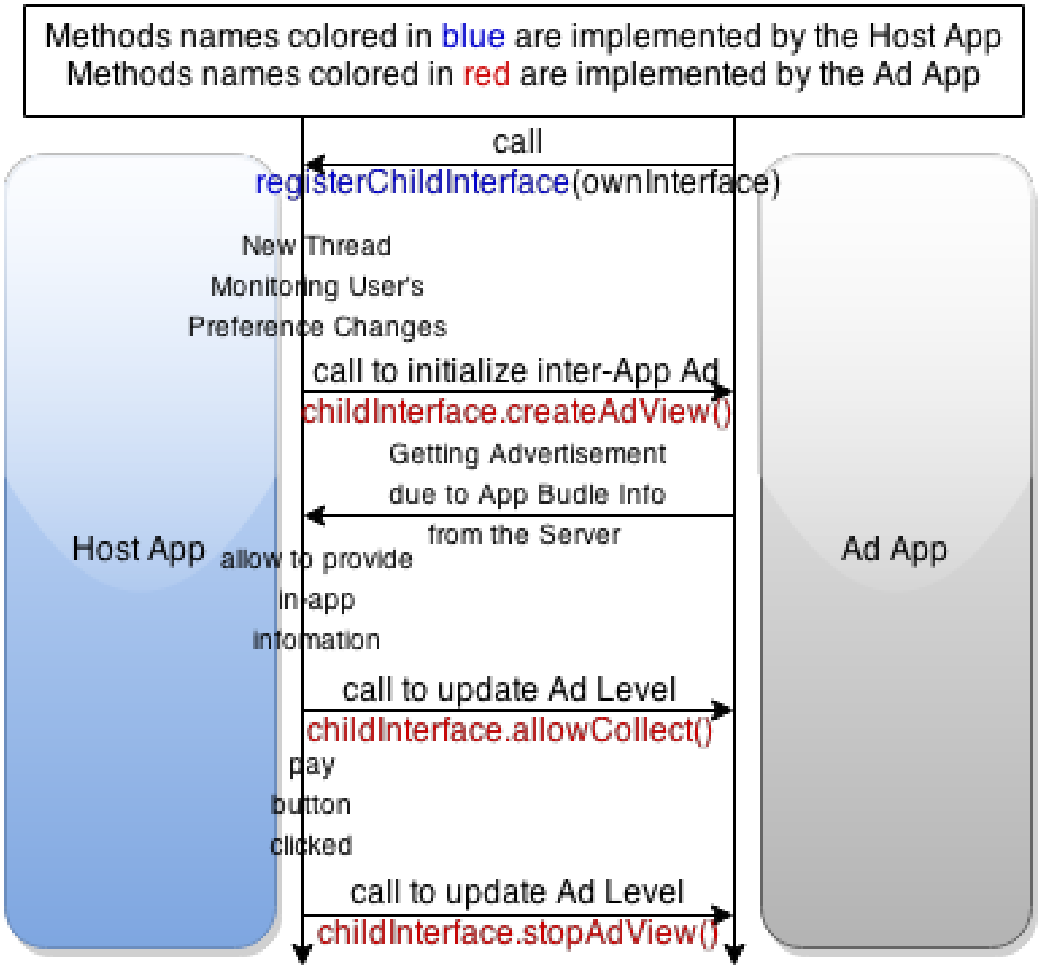

Methods names colored in blue are implemented by the Host App
Methods names colored in red are implemented by the Ad App
call
registerChildInterface(ownInterface)
New Thread
Monitoring User's
Preference Changes
call to initialize inter-App Ad
childInterface.createAdView()
Getting Advertisement
due to App Budle Info
from the Server
Host App
allow to provide
in-app
infomation
Ad App
call to update Ad Level
childInterface.allowCollect()
pay
button
clicked
call to update Ad Level
childInterface.stopAdView()

Figure 6.3: Communication between Host App and Ad App process

and asks the user's permission for the advert to be fully displayed. Once the user agrees to view the whole advertisement, the ad application starts an new *Intent* to call the system browser. The browser brings out the complete information about the application in the displayed advertisement by loading contents using the ad's corresponding url. On the page loaded, the user can choose to read the full description of the application and download it.

The inter-process communication is accomplished using *Android interface definition language*. In a word, both the host application and the ad application define interfaces and claim methods and variables which are available for the other one to use. The AIDL compiler outputs an interface translated as the Java programming language from the interface defined in the *.aidl* file. Then both application implement their own interface methods which are necessary for IPC calls. An inner abstract class named *Stub* that inherits the interface is implemented in both the host and the ad applications.

Figure 6.3 shows the procedure of how the embedded ads are shown and updated. The host application implements *IEmbeddedContainer* while the ad application implements *IRemoteEmbeddedAd*. When the host application is first created and opened, the *onCreate* method is called and everything is initialized. Then the host application shares an instance of the *IEmbeddedContainer* to the child process (here the ad application) and refers it as *parentBinder*. The ad application creates an instance *ownInterface* of the *IRemoteEmbeddedAd* and in the meantime gets the reference to *parentBinder*. The child process then passes its *ownInterface* as an argument to the main process via *containerInterface.registerChildInterface(ownInterface)*, which is implemented in the *Stub* class in the host application. The host application starts a new thread to monitor user preference. When the user allows the collection of his demographic information, the host application sends the information to ad application, which then communicates with the ad library server. The user gets more targeted ads after the upload of demographic information, and also unlocks the professional functionality. On the other hand, when the user chooses to make a payment to the provider directly, the third functionality could also be unlocked and the user gets rid of all the advertisements.

The communication actions with the ad library server are all implemented in the ad application using Socket. For consistency the ad application is using the same *Transfer* class as is used on the server side. When it starts to send a command, it first uses *Transfer.FromStringToBytes()* to interpret the command into bytes before communicating with the server. As is mentioned in section 6.1, the data sent from this Client side also strictly follows the loop of sequences of *Command number–Data length–Data–Command number–...*.

## 6.3 Advertising Procedure

After the illustration of the details on the ad library server side and the application client side, we now come to an overview of the complete procedure of our advertising, which is shown in Figure 6.4.

We set up our multi-thread server to wait for the connection request from the Android client. The client sends command ID 1 using *socket.write()* to request a connection. After a connection is successfully established, the client then starts to send App Bundles information to the server using command ID 2, followed by the App Bundles list size, and then, length of each package name, and finally the packagename. All the strings are translated into bytes before goes to Socket.

After finishing receiving all the data after command 2, the server first translates the bytes into strings and selects appropriate ads due to the app types from the database. A collection $C$ of three lists which all follow the same index order—AdPicLocation list, AdDownloadURL list, UserAgeRange list—is then created due to the result set of querying the database. When user's age information has not been updated from the client, the server sends periodically (refreshing every 10 seconds in our Demo) both the advertisement and the corresponding downloading url of every application in $C$ to the client side.

Similar to command ID 2, here the server sends command ID 3 to the client to indicate the start of actual advertising. For each advertisement in the list, the server sends the length of the ad picture file, then the file itself, followed by the length of the download url, and finally the download url. Once the user selects to exchange his demographic information for more functionality, the server further filters the ads (stop sending those Ads that the age range does not match with the user of the current host application) and keeps refreshing. The user can also choose to get more functionality by making a payment directly to the application provider and stop receiving ads

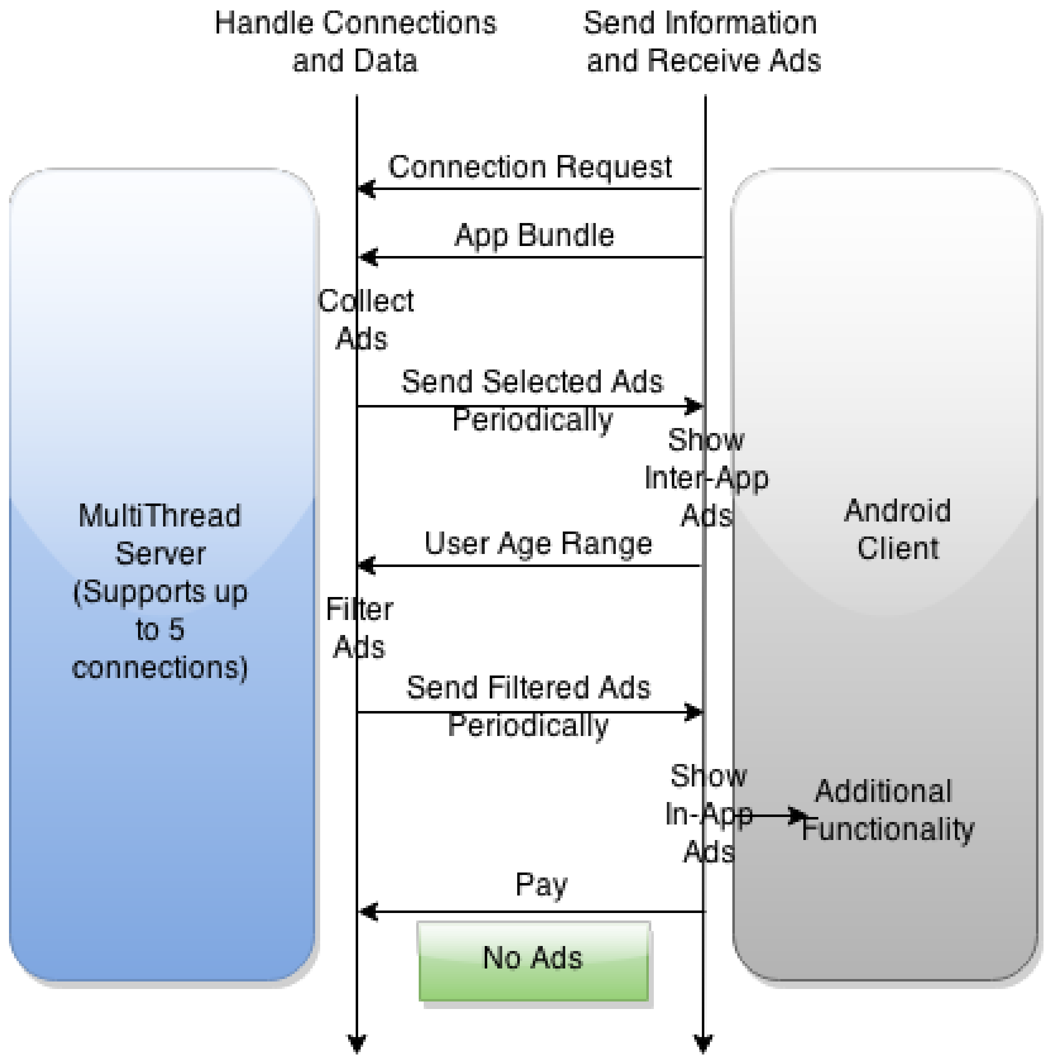

Handle Connections and Data
Send Information and Receive Ads
Connection Request
App Bundle
Collect Ads
Send Selected Ads Periodically
Show Inter-App Ads
MultiThread Server (Supports up to 5 connections)
Android Client
User Age Range
Filter Ads
Send Filtered Ads Periodically
Show In-App Ads
Additional Functionality
Pay
No Ads

Figure 6.4: Communication process between Ad Library Server and Android Client

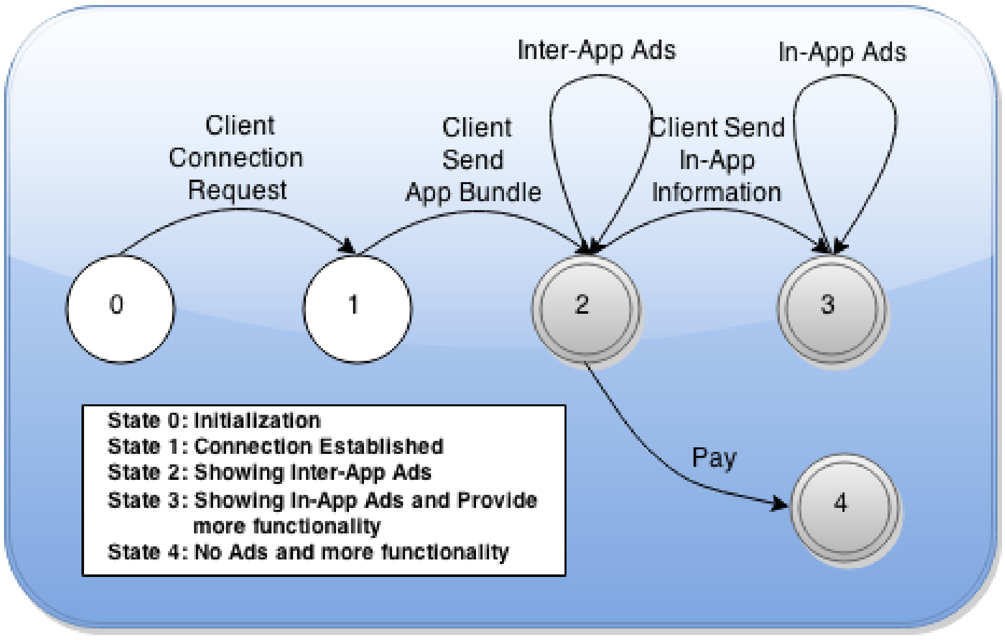

Inter-App Ads
In-App Ads
Client Connection Request
Client Send App Bundle
Client Send In-App Information
0
1
2
3
Pay
4
State 0: Initialization
State 1: Connection Established
State 2: Showing Inter-App Ads
State 3: Showing In-App Ads and Provide more functionality
State 4: No Ads and more functionality

Figure 6.5: Advertising Process and State Transition Finite State Automaton

from the advertiser.

**Algorithm 1** Advertisement Selection and Filtering Algorithm

```
1: procedure ADVERTISING
2:     Client gets user age $a$ from host app and sends to server
3:     for each app name $n \in$ App Bundle $L$ do
4:         Client sends $n$ message to Server
5:         Server queries database Ad table using $n$ and get the app type $t$
6:         Server queries database Ad table using $t$ and get result set $S$
7:         for each result $s \in$ result set $S$ do
8:             add ad location to list $L$
9:             add ad url to list $U$
10:            add app user age range to list $R$
11:        end for
12:        for each age range $ar \in R$ do
13:            if $a$ matches $ar$ then
14:                get ad from ad location and send to Client
15:                get ad url and send to Client
16:            end if
17:        end for
18:    end for
19: end procedure
```

One thing worth noting here is that, once the actual advertising has started, the server is only sending ads and the client is only receiving ads. Socket supports full-duplex communication so the input stream of the server side and the output stream of the client side can still be utilized. In our implementation, the server has a child thread which monitors the reporting of user age from the client, while on the client side, the ad application has a child thread that listens to the host application, once the user allows the sharing of age information in the host application, this child thread reports it to the server.

Figure 6.5 uses a finite state automaton to describe the state transitions of the above procedure. State 0 indicates the initialization of the system, state 1 shows the connection of the server and the client, when in state 2, the user keeps receiving Inter-App Ads until he chooses any of the two options to upgrade his Health Coach Master to get more functionality, while sharing in-app information (user's age in our case) leads to state 3, and payment leads to state 4. A user cannot transit from state 3 to state 4 or in the opposite direction.

Algorithm 1 introduces how the advertisement library server selects and filters ads with regard to a particular client. Before the user choose to share the age information with the server, the server simply collects and collates all the possible advertisements matching the type of every application from App Bundles information. After age information is uploaded, it is used as a filter to get rid of any application that does not match the reported age range.

# CHAPTER 7

# DEMO AND EVALUATION

After the implementation of our model, we do a demo using the ad library server and the Android application client. Section 7.1 shows the procedure of how the implemented demo works. Section 7.2 demonstrates the advantages of our work and provides analysis of limitations of this work.

## 7.1 Demo

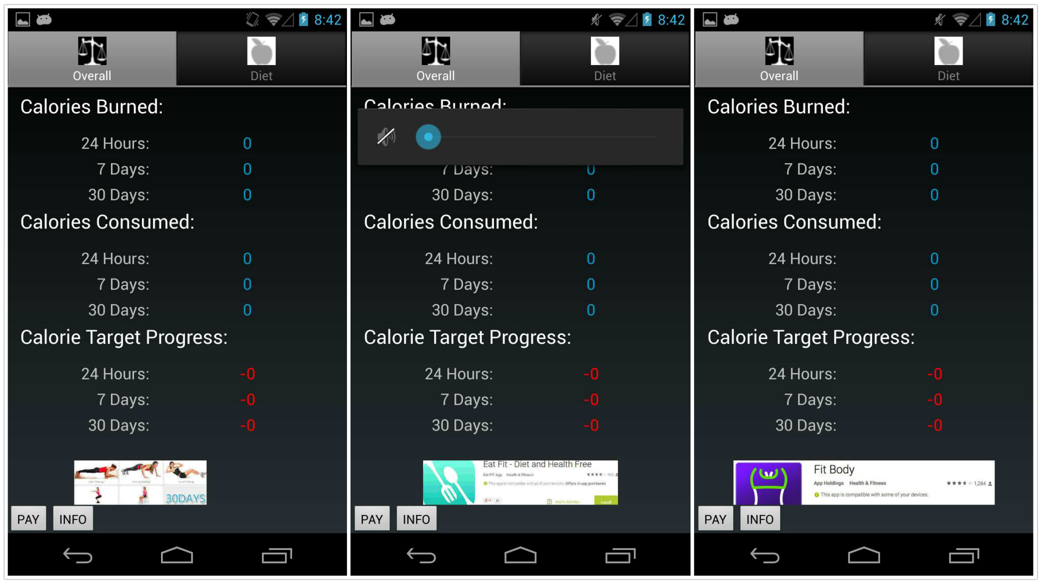
We use 8 sample applications in our demo. Table 7.1 shows the information of the related applications:

Table 7.1: Advertisements Used in the Demo

| ID | AppName | AdPicLoc | AppUrl | AppType | UserAgeRange |
|---|---|---|---|---|---|
| 0 | $30DAYS$ | ... | ... | 0 | 2X |
| 1 | $EATFIT$ | ... | ... | 0 | 3X |
| 2 | $FITBODY$ | ... | ... | 0 | 4X |
| 3 | $LAYAMUSIC$ | ... | ... | 1 | 2X |
| 4 | $MUSICPLAYER$ | ... | ... | 1 | 3X |
| 5 | $BEATSMUSIC$ | ... | ... | 1 | 4X |
| 6 | $HealthCoachManager$ | ... | ... | 0 | 2X |
| 7 | $Music$ | ... | ... | 1 | 3X |

First we will explain what does each of the keys in Table 7.1 mean.

- *ID* is the unique primary key of each application in the database, it starts from 0 and it is auto-increment.



Calories Burned:

| | |
|---|---|
| 24 Hours: | 0 |
| 7 Days: | 0 |
| 30 Days: | 0 |

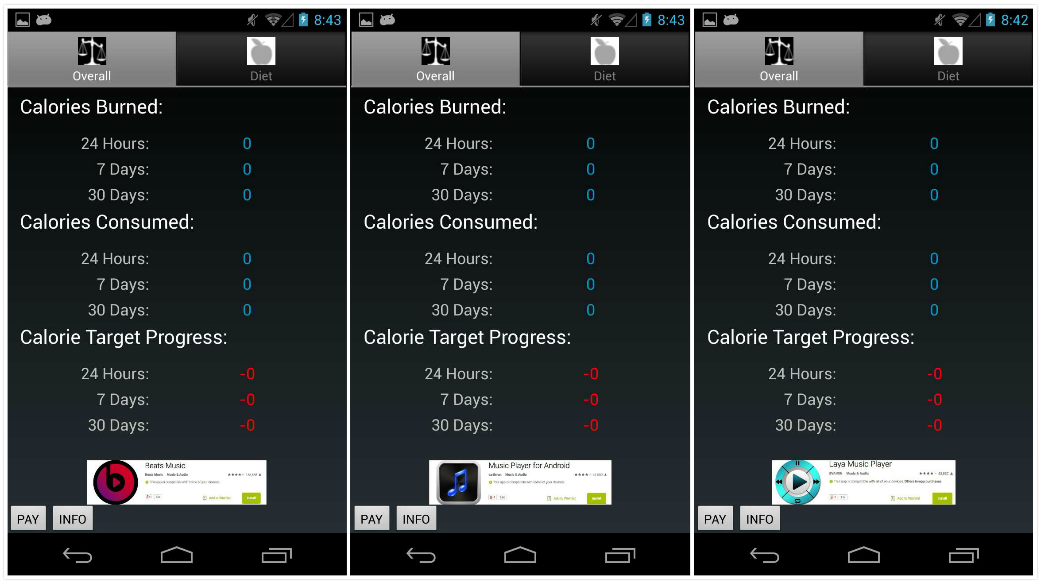
Calories Consumed:

| | |
|---|---|
| 24 Hours: | 0 |
| 7 Days: | 0 |
| 30 Days: | 0 |

Calorie Target Progress:

| | |
|---|---|
| 24 Hours: | -0 |
| 7 Days: | -0 |
| 30 Days: | -0 |

Figure 7.1: Ads Loop 1 : Health-related (AppType 0) Ads



Calories Burned:

| | |
|---|---|
| 24 Hours: | 0 |
| 7 Days: | 0 |
| 30 Days: | 0 |

Calories Consumed:

| | |
|---|---|
| 24 Hours: | 0 |
| 7 Days: | 0 |
| 30 Days: | 0 |

Calorie Target Progress:

| | |
|---|---|
| 24 Hours: | -0 |
| 7 Days: | -0 |
| 30 Days: | -0 |

Figure 7.2: Ads Loop 2 : Music-related (AppType 1) Ads

- *AppName* shows the packagename the same as what could be got using *getInstalledPackage()* by our Android application client.
- *AdPicLoc* is the locations of the advertisement picture stored in the Server's local disk.
- *AppUrl* is the url in Google Play Store which leads to the downloading page of the advertised application.
- *AppType* shows which AppType does a application belong to. In our demo we're using 2 AppTypes of applications. When the value of AppType is 0, it means the application is health-related, while AppType 1 means it is a music-related application.
- One application is always most prevalent among a particular user group in a certain age range, revealed by *UserAgeRange* in the above Table.

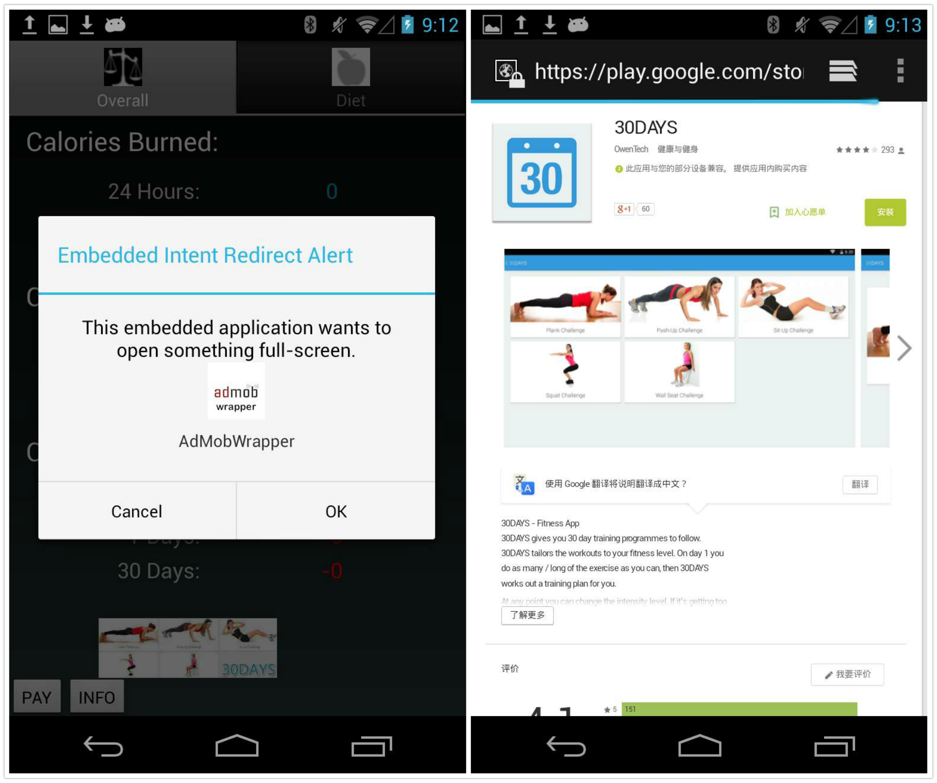
The 8 sample ads include 4 apps in AppType 0 and 4 apps in AppType 1, 3 apps most popular in the user group of ages 20s, 3 most prevalent with users in their 30s and 2 for those in their 40s.

Our demo host application *Health Coach Manager* is a health-related application, its *AppType* is recorded as 0 in Table 7.1. Other installed applications in our test device include *Music, Bluetooth, Email, Browser, etc.*

The procedure of advertising is as follows.

When the Health Coach Master application is first opened, it is initialized and an EmbeddedView instance is created. Now the user is at *Inter-App Ads* Level and get basic functionality of the application: the overall information and the UI of recording his calories taking from meals. The embedded ad application sends App Bundles information to the ad library server. The list of package names in our App Bundles information includes Music, Bluetooth, Email, Browser, Health Coach Manager, etc. Only 2 of them are in our test data set, Music and Health Coach Manager, of AppType 1 and 0 respectively. The querying of *AppType = 0* returns a set of applications: *30DAYS, EATFIT and FITBODY*. Similarly, The querying of *AppType = 1* returns *LAYAMUSIC, MUSICPLAYER and BEATSMUSIC*. The server then is prepared with a list of applications: *30DAYS, EATFIT and FITBODY, LAYAMUSIC, MUSICPLAYER and BEATSMUSIC*. The server then sends ads to the client, one at a time, and refreshes every 10 seconds. The refreshments of advertisements are shown in Figure 7.1 and Figure 7.2.

Clicking on any of these ads prompts a dialogue asking the user's permission to display the complete ad in a browser, shown in Figure 7.3. Once the user chooses "OK" the application starts a new intent to call the browser, in which the application is completely shown in Google Play Store, where the user can view the full information of the app and install it.



Embedded Intent Redirect Alert

This embedded application wants to open something full-screen.

AdMobWrapper

Cancel OK

此应用与您的部分设备兼容。 提供应用内购买内容

使用 Google 翻译将说明翻译成中文？

30DAYS - Fitness App
30DAYS gives you 30 day training programmes to follow.
30DAYS tailors the workouts to your fitness level. On day 1 you do as many / long of the exercise as you can, then 30DAYS works out a training plan for you.
At any point you can change the intensity level. If it's getting too

Figure 7.3: Click on Ads to Show Application in Google Play Store

The two buttons on the bottom of the host application, *PAY* and *INFO*, which can also be seen in Figure 7.1 and Figure 7.2 provide the user opportunities of upgrading his service.

When the *Pay* button is clicked the user pays service fees directly to the application developer to unlock the third functionality: keep track of calories consumed via exercises, and also gets rids of all the advertisements. The result of the payment activity is shown in Figure 7.4.

The other button *Info*, when clicked, it means the user agrees to share his demographic information with the advertiser to unlock the *Exercise* function-

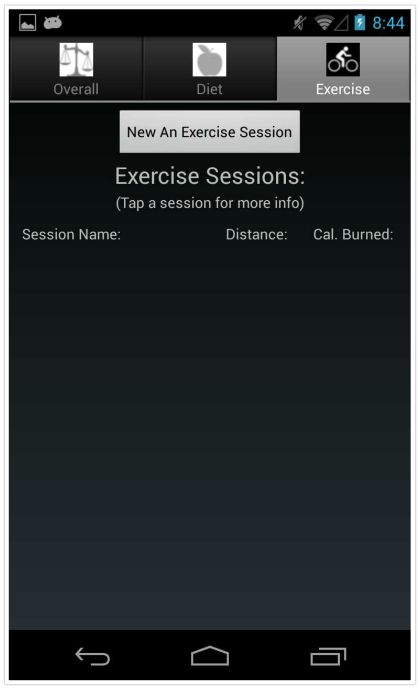


Figure 7.4: Unlock Professional Functionality via Payment

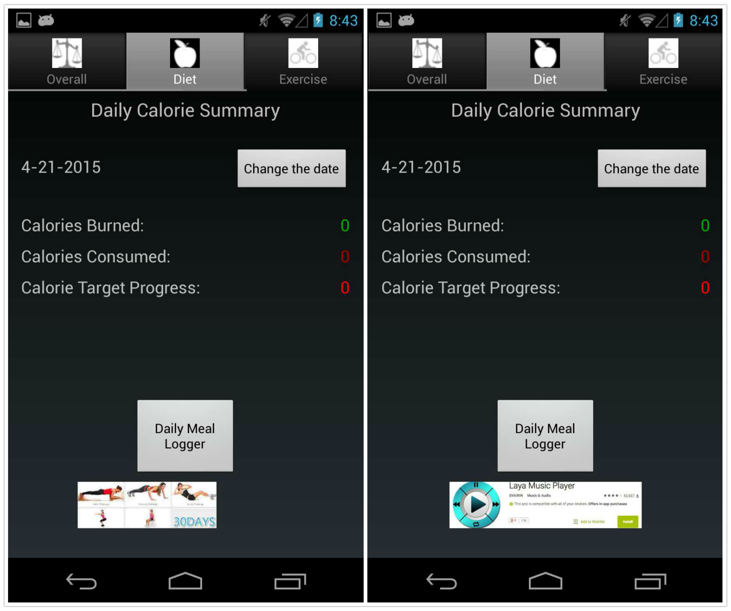


Figure 7.5: Unlock Professional Functionality via Payment

ality and gets more targeted ads. In our demo, our host application collects the user's age information, 20, and updates it to the server. The server, which has been monitoring the receiving of the user's demographic information, starts to use *UserAge* as a filter to stop sending less matching ads. In Table 7.1, only 30DAYS and LAYAMUSIC are recorded as most prevanlent among the users in their 20s, so the resulting advertisements are shown in Figure 7.5, only the two ads for 30DAYS and LAYAMUSIC are periodically delivered to the client.

## 7.2 Evaluation and Analysis

From our demo we demonstrate that our model effectively provides permission separating for the advertisement module and the host application. The design of privacy levels for the users to choose considers the preferences of different users and balances between service and privacy.

The advantage of our approach is obvious: first of all, our model solves the problem of privacy offense with regard to the user; on the other hand, the benefits of the advertiser is also protected by preventing click fraud cheating from the provider; last but not least, for the provider, the ad library is no longer able to abuse the permissions it gets from the host application. In a word, the three party feedback loop balances the benefits of each entity in the advertisement model. There are actually some applications on the iOS platform that provides in-app purchases [35] to get rid of the ads. This is a "two-layer" advertisement model to some extent, providing a *No Ads* level and a *Having Ads* Level. Our work, designed with four layers, considers more about the variances and changes of users' preferences by offering the users with more levels to choose.

There are, however, limitations of our work. Most significantly, the deployment of LayerCake system as a platform means that our model does not work in any original Android system. It is difficult for Google to incorporate the new model into their design since the current model is still prevalent and has not caused too many complaints from the users. What's more, our model proposes that all the permissions required by different applications being clearly stated to the user, which might be good to some users who have good security and privacy background knowledge, but might also cause panic among those who have not been caring too much about their privacy previously. This means our model could also be hard for the users to accept. Also, our work uses a simple and naive recommendation algorithm: find applications which are similar to what have been installed, which means our prediction may not be accurate enough.

# CHAPTER 8

# CONCLUSION AND FUTURE WORK

In this master thesis work, we study the privacy problems in the current Android on-line advertising model. With the explore of the background knowledge, we discover that many Android applications rely on the revenue from the advertiser. The current advertising model brings about privacy offense problems with regard to the user. After doing the analysis of know attempts of preserving user privacy, we address two approaches in our work. Our first approach separates ad library permission requirements from the host application and forms a clear three party feedback loop, and the second approach provides the user with a four-level privacy concerned advertisement mechanism.

Our model call into further attention to alternate the design of current Android on-line advertising model. However, our design still has some limitations. First, our implementation uses LayerCake System as a platform, which requires the modification of the Android system source code. This lead to the difficulty for Google to incorporate the new model into their design in recent years. Second, our new model proposes to use App Bundles information and presents the user with clear described permission requirements. This could be good news to some users who have background knowledge about privacy but could also cause panic among the other users who have not been caring anything about their personal information previously. In the latter case, the new model would actually be even harder to be accepted, before the users understand why and how their privacy should be protected. Third, our advertising procedure adopts a naive recommendation and prediction algorithm, and has been doing the test using a relatively small data set.

There is still a long way to go before this work can be applied in real life. There are some future works that could be done. First, we can design and adopt more reasonable and accurate prediction algorithms for the advertisement selection. Second, we are using static App Bundle information in our current implementation, however, we could always modify our code to support App Bundles information updating, which means, whenever after the user installs a new application, the ads targeted at him may be different.

# REFERENCES

[1] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, “Mockdroid: trading privacy for application functionality on smartphones,” in *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications.* ACM, 2011, pp. 49–54.

[2] M. Nauman, S. Khan, and X. Zhang, “Apex: extending android permission model and enforcement with user-defined runtime constraints,” in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security.* ACM, 2010, pp. 328–332.

[3] D. Quercia, I. Leontiadis, L. McNamara, C. Mascolo, and J. Crowcroft, “Spotme if you can: Randomized responses for location obfuscation on mobile phones,” in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on.* IEEE, 2011, pp. 363–372.

[4] M. P. I. Leontiadis, C. Efstratiou and C. Mascolo., “Dont kill my ads! balancing privacy in an ad-supported mobile application market.” in *12th Workshop on Mobile Computing Systems and Applications, San Diego, CA, HotMobile 12*, 2012.

[5] F. Roesner and T. Kohno., “Securing embedded user interfaces: Android and beyond.” in *USENIX Security*, 2013.

[6] “Android open source project. dex http://source.android.com/tech/dalvik/dex-format.html,” 2007.

[7] J. MARSHALL, “Twitter is tracking users installed apps for ad targeting,” 2014. [Online]. Available: http://blogs.wsj.com/cmo/2014/11/26/twitter-is-tracking-users-installed-apps-for-ad-targeting/

[8] A. Henry, “Facebook is tracking your every move on the web; here’s how to stop it,” 2011. [Online]. Available: http://lifehacker.com/5843969/facebook-is-tracking-your-every-move-on-the-web-heres-how-to-stop-it

[9] R. Agrawal and R. Srikant, “Mining sequential patterns,” in *Data Engineering, 1995. Proceedings of the Eleventh International Conference on.* IEEE, 1995, pp. 3–14.

[10] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2001, pp. 0215–0215.

[11] Google, "Admob ads sdk." [Online]. Available: https://developers.google.com/mobile-ads-sdk/

[12] M. D. Shekhar, Shashi and D. S. Wallach., "Adsplit: Separating smartphone advertising from applications." in *USENIX Security Symposium*, 2012.

[13] S. S. Y. P. A. S. Dietz, Michael and D. S. Wallach., "Quire: Lightweight provenance for smart phone operating systems." in *USENIX Security Symposium*, 2011.

[14] F. Roesner, J. Fogarty, and T. Kohno, "User interface toolkit mechanisms for securing interface elements," in *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 2012, pp. 239–250.

[15] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan, "User-driven access control: Rethinking permission granting in modern operating systems," in *Security and privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 224–238.

[16] A. NEWCOMB, "Twitter is watching you: How to opt out of new tracking feature," 2014. [Online]. Available: http://abcnews.go.com/Technology/twitter-watching-opt-tracking-feature/story?id=27204927

[17] K. Wagner, "Twitter to start tracking which apps its users have downloaded," 2014. [Online]. Available: http://recode.net/2014/11/26/twitters-now-collecting-data-on-which-apps-you-download/

[18] E. M. Zeman, "Twitter now tracking which apps you've installed," 2014. [Online]. Available: http://www.phonescoop.com/articles/article.php?a=14964

[19] "Get search, twitter api," 2013. [Online]. Available: https://dev.twitter.com/docs/api/1/get/search

[20] Wikipedia, "Twitter now tracking which apps you've installed," 2015. [Online]. Available: http://en.wikipedia.org/wiki/Software_development_kit

[21] L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schecter, and C. Jackson, "Clickjacking: Attacks and defenses." in *USENIX Security Symposium*, 2012, pp. 413–428.

[22] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, “A survey of mobile malware in the wild,” in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices.* ACM, 2011, pp. 3–14.

[23] A. BRAIN, “Discover new android apps, manage your apps, share your apps with friends.” [Online]. Available: http://www.appbrain.com/

[24] V. Agrawal, “App usage tracer.” [Online]. Available: https://play.google.com/store/apps/details?id=com.agrvaibhav.AppUsageTracking&hl=en

[25] Google, “Google play,” 2015. [Online]. Available: https://play.google.com/store/search?q=traffic+monitor&c=apps

[26] R. Srikant and R. Agrawal, *Mining sequential patterns: Generalizations and performance improvements.* Springer, 1996.

[27] J. Reps, J. M. Garibaldi, U. Aickelin, D. Soria, J. E. Gibson, and R. B. Hubbard, “Discovering sequential patterns in a uk general practice database,” in *Biomedical and Health Informatics (BHI), 2012 IEEE-EMBS International Conference on.* IEEE, 2012, pp. 960–963.

[28] A. P. Wright, A. T. Wright, A. B. McCoy, and D. F. Sittig, “The use of sequential pattern mining to predict next prescribed medications,” *Journal of biomedical informatics*, 2014.

[29] L. Ulanoff, “Amazon knows what you want before you buy it,” 2015. [Online]. Available: http://www.predictiveanalyticsworld.com/patimes/amazon-knows-what-you-want-before-you-buy-it/

[30] R. Barnes, “Amazon’s ’anticipatory shipping’ explained,” 2014. [Online]. Available: http://www.marketingmagazine.co.uk/article/1228379/amazons-anticipatory-shipping-explained

[31] “All about cookies.” [Online]. Available: http://www.allaboutcookies.org/cookies/cookie-profiling.html

[32] H. Haddadi, P. Hui, and I. Brown, “Mobiad: private and scalable mobile advertising,” in *Proceedings of the fifth ACM international workshop on Mobility in the evolving internet architecture.* ACM, 2010, pp. 33–38.

[33] J. Camenisch, D. Sommer, S. Fischer-Hübner, M. Hansen, H. Krasemann, G. Lacoste, R. Leenes, J. Tseng et al., “Privacy and identity management for everyone,” in *Proceedings of the 2005 workshop on Digital identity management.* ACM, 2005, pp. 20–27.

[34] F. F-Secure Labs, Helsinki, “Mobile threat report,” 2013. [Online]. Available: https://www.f-secure.com/documents/996508/1030743/Mobile_Threat_Report_Q3_2013.pdf

[35] CodeTuition, “Integrating ios app with in-app purchase,” 2013. [Online]. Available: http://www.codetuition.com/ios-tutorials/integrating-ios-app-with-in-app-purchase/