NTNU – Trondheim
Norwegian University of
Science and Technology

Enabling Energy-Efficient Advertising for Mobile Applications

Irena Prochkova

Master in Security and Mobile Computing
Submission date: June 2013
Supervisor: Yuming Jiang, ITEM
Co-supervisor: Jukka K.Nurminen, Aalto University

Norwegian University of Science and Technology
Department of Telematics

Aalto University
School of Science
Degree Programme in Security and Mobile Computing

Irena Prochkova

Enabling Energy-Efficient Advertising for Mobile Applications

Master's Thesis
Espoo, June 30, 2013

Supervisors: Professor Jukka K. Nurminen, Aalto University School of Science
Professor Yuming Jiang, Norwegian University of Science and Technology
Instructor: Jukka K. Nurminen

Aalto University
School of Science
Degree Programme in Security and Mobile Computing

ABSTRACT OF
MASTER'S THESIS

Author: Irena Prochkova

Title:
Enabling Energy-Efficient Advertising for Mobile Applications

Date: June 30, 2013 **Pages:** 64

Professorship: Data Communication Software **Code:** T-110

Supervisors: Professor Jukka K. Nurminen
Professor Yuming Jiang

Instructor: Jukka K. Nurminen

Advertisements are the main source of revenue for many free mobile applications, however, they increase the energy consumption of the mobile device. In particular, the radio communication used for the advertisement data transfer is energy hungry, so advertisement sponsored applications (free) consume more energy than paid applications.

In this thesis, we analyse the effect that advertisements have on the mobile device performance, especially, the energy consumption of transferring and displaying advertisements. Our results, based on evaluating 5 mobile games, show that advertisements consume up to ≈25% of application's total power consumption.

To make advertising energy-efficient, we propose and implement a solution (*eeAdNetwork*) that reduces energy consumption by intercepting advertisement requests and serving them from a local cache. The proposed solution acts as an extension to the AdMob advertising network. We also run measurements to evaluate the performance of our proposed solution. Our results show that the power consumption of delivering and displaying advertisements is decreased from 27% to 4% of the application's total power consumption.

Keywords: advertising, energy-efficiency, mobile advertising, AdMob, power consumption, caching, caching strategy

Language: English

Acknowledgements

The thesis would not have been completed without the help and support of several people. I wish to thank my supervisor and instructor Jukka Nurminen for his timely instructions and guidance before and during the thesis. I am especially appreciative for his positive thinking and optimism that helped me complete this task. I would also like to thank my other supervisor, Yuming Jiang for providing thoughtful comments and feedback on the thesis.

I also appreciate the ideas, comments and feedback from my colleagues and friends at Aalto University, Department of Data Communications Software. My time at Aalto University and NTNU would not have gone so smoothly without the help of the NordSecMob co-ordinators: Anna Stina Sinisalo, Mona Nordaune and Misela Väisänen.

Finally, I thank my family for their love, care and support.

Espoo, June 30, 2013

Irena Prochkova

Abbreviations and Acronyms

3G	Third Generation
API	Application Program Interface
CPA	Cost per Action
CPC	Cost per Click
CPM	Cost per Thousand Impressions
CPU	Central Processing Unit
CSS	Cascading Style Sheet
CSV	Comma Separated Value
DNS	Domain Name Service
GPRS	General Packet Radio Service
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IPTABLES	Internet Protocol Tables
MMA	Mobile Marketing Association
MMS	Multimedia Messaging Service
OS	Operating System
RAM	Random Access Memory
REST	Representational State Transfer
SDK	Software Development Kit
SMS	Short Message Service
TTL	Time To Live
TV	Television
UTF8	Unicode Transformation Format (8-Bit)

Contents

Chapter 1

Introduction

The number of smartphone users (≈1 billion [15]) have increased significantly over the last few years. The explosive adoption rate has been driven by a number of factors, with the most important influence being the emergence of mobile applications (apps) and application markets. Mobile applications are software applications that run natively on smartphones, tablet computers and other mobile devices. Application markets on the other hand are digital distribution platforms for applications that allow users to browse and download application on their mobile devices. Example of application markets are: Google Play[1], Apple's App Store[2] and Windows Phone Store[3]. The application market mainly publishes two types of mobile applications on: *free* and *paid* applications. Free applications are downloaded free of cost but use advertisements to generate revenue, while paid applications are purchased for a fixed price and do not show advertisements. Currently, free applications account for 90% of all applications on the application markets [10], which gives the impression that advertising is the preferred form of monetization for mobile applications.

Mobile advertising is defined as a form of marketing through mobile devices, that promotes products, services, or businesses to the end-users. Advertisements in mobile devices can be displayed inside mobile web pages,

[1]https://play.google.com/store?hl=en
[2]http://www.apple.com/iphone/from-the-app-store/
[3]http://www.windowsphone.com/en-us

mobile applications or even through Short Message Service (SMS) messages. Mobile applications usually display advertisements as graphical banners (image and text) at the top or bottom of the screen and the advertisement content is downloaded from an advertising network's server (for example, AdMob[4]). The current advertisement delivery process requests advertisements at fixed intervals (every 30, 60, 90, or 120 seconds), which involves frequent data transfers over the network that increases the energy consumption of the mobile device [2]. Consequently, fetching advertisements at short intervals requires an always-on Internet connection that some applications do not necessarily require. For example, mobile arcade games (e.g., Angry Birds[5]) do not require Internet access and only use the Internet to receive advertisements.

While managing energy consumption in desktop computers is not very important, in mobile devices it is crucial for extending the mobile phone's battery life. Unfortunately, mobile phone's battery life have not advanced as rapidly as the other mobile components [1], therefore, we need to make sure that applications utilise the battery in an energy-efficient manner. We find that applications that only use the Internet to fetch advertisements, in particular mobile games, can be optimised to deliver advertisements more efficiently. In this thesis, we propose, implement and evaluate an energy-efficient approach of delivering and displaying advertisements in mobile applications.

1.1 Objectives and Research Goals

The main goals of the thesis are to: 1) study the current mobile advertising ecosystem and analyse the interaction between the advertising parties, 2) measure the energy consumption of delivering advertisements using current methods in mobile applications, 3) propose and implement an energy-efficient solution for delivering advertisements, and 4) evaluate the performance of our proposed solution.

[4]http://www.google.com/ads/admob/
[5]http://www.angrybirds.com

1.2 Author's Contribution

Mobile advertising is relatively new area that requires interpretation with a broad view. The mobile advertisement eco-system is rather complex, consisting of many components (such as brands, advertising agencies, advertisers and publishers), each of them working together to deliver advertisements to the end users. In this thesis, the objective is to study the mobile advertising eco-system from broad a perspective, including all forms of advertising and all components involved in the process.

In this thesis, I analysed the effect of advertising on mobile phone's battery, by running experiments to measure the power consumption of delivering and displaying advertisements. My experiments show that the existing methods of delivering advertisements are not energy-efficient.

To decrease the energy consumption, I developed and implemented a caching solution, *eeAdNetwork*, that works as a middleware between the mobile application and the advertising network. It is backwards compatible and works within the constraints posed by the mobile operating system and the advertising platform.

I evaluate the performance of the proposed solution and my results show a decrease in energy consumption. Furthermore, in my evaluation, the application fetches advertisements over both 3G and WiFi networks. It should be noted that preliminary results have been published in [27].

1.3 Structure of the Thesis

The thesis is structured as follows. After the introduction, in Chapter 2 we discuss the mobile advertising eco-system and the process of delivering advertisements in mobile application. We also state the main concerns of mobile advertising. Chapter 3 analyses the impact of advertising on mobile phone's battery. This includes a thorough literature review and initial experiments that measure the energy consumption of fetching advertisements in mobile applications. Chapter 4 is the design and the implementation of our solution, while Chapter 5 evaluates the proposed solution in multiple scenarios

and different caching strategies. Chapter 6 contains an extensive discussion and Chapter 7 concludes the thesis.

Chapter 2

Mobile Advertising

The mobile advertising ecosystem is a complex system that consists of different entities (e.g., brands, advertising agencies, advertising networks and developers) that works together to deliver advertisements to the end-user. In this chapter, we discuss about the characteristics of the different entities in the mobile advertising ecosystem, the advertisement delivery process, and the main challenges which affect mobile advertising, such as, privacy, security and energy concerns.

2.1 The Mobile Advertising Ecosystem

The mobile advertising ecosystem consist of the following entities [34]: *Brands*, *Agencies*, *Ad Networks*, *Publishers* and *Consumers*.

Figure 2.1: Mobile advertising ecosystem.

Brands: are advertising companies that want to promote products or services to the consumers, such as Nike, Coca Cola, McDonalds and Disney.

Brands can also be small unknown companies that choose to advertise their products on mobile devices because of its easy usage, compared to print advertising.

Agencies: are marketing companies that create advertising campaigns and strategies for the *brands*.

Ad Networks: are companies that interlink the *brand* and the *publisher*. The *Ad Network* stores advertisements created for the *brands* and sends them to the appropriate *publisher* for presentation. Example of *Ad Networks* are: AdMob and iAd[1]. The characteristics of the *Ad Networks* and their relationship are discussed in 2.1.3.

Publishers: are developers of mobile applications that integrate the advertisement in their applications. This is similar to Internet advertising where the advertisements are embedded into the web site. The *publishers* are motivated to integrate the advertisement because it provides a way to monetize the application/service.

Consumers: are the end-users that engage with the mobile application and interact with the advertisement.

Figure 2.1 shows the workflow in the mobile advertising ecosystem. Typically, *brands* initiate the process by collaborating with an *advertising agency* to create the advertisement. To distribute the advertisement over the mobile Internet, the *agency* or the *brand* approaches an *Ad Network* which provides the platform (or service) to distribute the advertisement to a targeted audience. The *developers* integrate the advertisements from the *Ad Network* into their application, which are consumed by the end-users.

2.1.1 Types of Mobile Advertisements

Mobile advertisements differ from Internet advertisements because they can be distributed not only via mobile webpages but can be integrated into native applications by the developers. Different types of mobile advertisements have emerged in the recent years, each of them using a specific distribution

[1]http://advertising.apple.com

channel in the mobile device. The Mobile Marketing Association (MMA)[2] categorises the different advertisements distribution channels into five units: Mobile Web, Text Messaging (SMS), Multimedia Messaging (MMS), Mobile Video and TV, and Mobile Applications [19].

Mobile Web allows delivery of web content to users via a mobile web browser. It is similar to PC-based Internet advertising, except that the advertisements have been tailored to meet the requirements of the mobile phone (for example, the screen size). Typically, they are presented as banner or textual advertisements. Banner advertisements are images or animations placed at the top or bottom of the mobile webpage. Text advertisements are text-based links embedded anywhere on the webpage. In both cases, a user can interact with the advertisement by clicking on it to obtain more information about the product/service being advertised. Figure 2.2 shows an example of a banner advertisement at the top of a webpage.

Text Messaging (SMS) allows users to communicate with one another using short messages, usually up to 160-characters (text). Some brands may use SMS to send advertisements to their consumers. Because of the wide range of mobile devices that can send and receive SMSes advertisers can reach a very large number of users. In some countries the consumer has to opt-in to receive these SMS advertisements. Figure 2.3 shows an example of an advertisement sent over an SMS.

Multimedia Messaging (MMS) is similar to an SMS, but instead of just textual information, it also contains rich multimedia (images, audio or video) content. Figure 2.4 shows an example of an image advertisement sent over MMS.

Mobile Video and TV is similar to the television advertisements, where video content is mixed with video advertisements, except in this case the content is adapted to the constraints of a mobile device (such as, screen size, memory). Like the television advertisements, the video advertisements in mobile video are embedded either at the beginning, in the middle, or at the end of the video content.

[2]it provides the technical specification for each type of mobile advertisement and also provides recommendations on advertisement creation and delivery process.

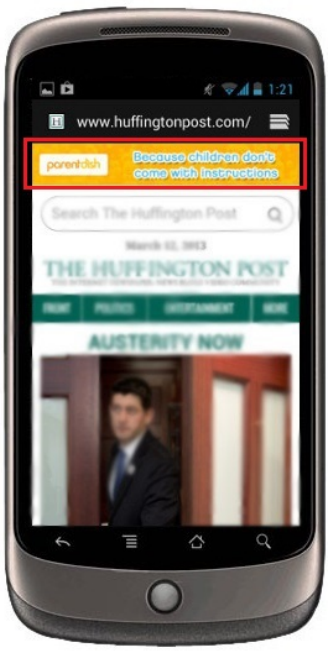


Figure 2.2: An example of mobile web advertisement

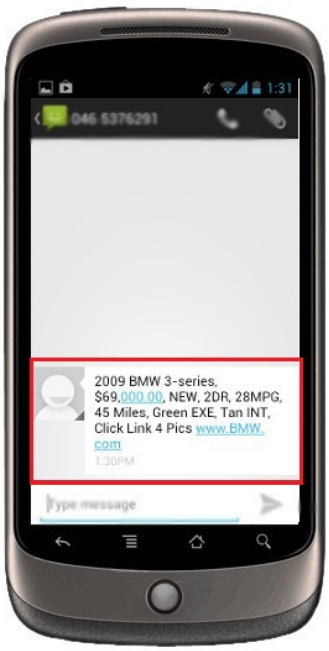


Figure 2.3: An example of SMS-based advertisement

Figure 2.4: An example of MMS-based advertisement

Mobile Application's advertisements are usually presented as a static image or rich media banner, similar to the one in web pages. Nowadays, many popular mobile games use in-app advertising as a method to monetize their applications (e.g Angry Birds on Android). Figure 2.5 shows an example of an in-app advertisement in a mobile game.

2.1.2 Pricing Models of Mobile Application

Mobile applications are sold in two ways: 1) directly by the developer's website, or 2) via an application market. With the emergence of the application markets, self publishing of applications by a developer has fallen out of favour in recent years (since 2009). The main reason is that the application market provides important services like, discovery (indexing and cataloging), signing (security), billing and in some cases verification of quality (by the app approval process). However, for these services, the application market takes

Figure 2.5: Angry Birds game displaying an advertisement in the bottom right corner.

a fraction ($\approx$ 30%) of the application sales. One major constraint for the application developer selling an application via application market is that it is priced once, i.e., the user buys the application and pays for it once, there is no method for subscription payments (monthly/yearly). Alternatively, the application is available for free of cost and the developer monetizes the application by displaying advertisements. In this case, the application developer shares the revenue (generated due to the engagement of the advertisement) with the *Ad Network*.

The basic pricing models for mobile applications available nowadays are: free applications (with advertisements), paid applications and in-app purchases (e.g., Books inside the Kindle application on an iPhone or additional levels/add ons in a game). Other ways to generate revenue from the application are freemium subscriptions (e.g., Spotify or Netflix), where the mobile application enables a service sold outside the application. Table 2.1 shows the share of free compared to paid applications' downloads. According to Gartner, Inc [10] free applications account for 90% of the downloads in 2013 which clearly indicates that free applications dominate the market. In terms of paid applications, 90% of the paid applications cost less than 3$ each. While the market is moving toward free and lower priced applications in-app purchase model should not be ignored. Gartner expects that the number of downloads featuring in-app purchase will increase from 5 percent of total

downloads in 2011 to 30 percent in 2016.

	2011	2012	2013	2014	2015	2016
Free Downloads	22,044	40,599	73,280	119,842	188,946	287,933
Paid Downloads	2,893	5,018	8,142	11,853	16,430	21,672
Free Downloads %	88.4%	89.0%	90.0%	91.0%	92.0%	93.0%

Table 2.1: Mobile App Store Downloads, Worldwide, 2010-2016 (Millions of Downloads). Source: Gartner [10].

For the rest of the sub-section, we discuss in detail the advertising-based pricing model. The mobile advertisements' pricing model is similar to the Internet-based advertising. There are generally three categories of pricing models [3]:

- CPM (Cost per Thousand Impressions): the *brand* pays the *Ad Network* every time the advertisement is displayed in any application.
- CPC (Cost per Click): the *brand* pays the *Ad Network* every time a user interacts with the advertisement. Typically, an event is counted as an interaction when the user clicks on the advertisement.
- CPA (Cost per Action): the *brand* pays the *Ad Network* every time a user engages in an action after interacting with an advertisement. Typically, this happens when a user buys the advertised product/service after seeing or clicking on the advertisement.

In practice the above pricing model works in the following way: the *brand* allocates a daily budget for each type of engagement (impressions, click, action) and it may prefer one over the others. The *Ad Network* takes the allocated money into account when displaing the advertisements and stops sending the advertisement to the application when any of the daily budget targets (CPM, CPC, CPA) are met.

In all the cases, the *Ad Network* shares the revenue with the associated application developer. For example, for each engagement (impression, click and/or action) on the advertisement the *Ad Network* keeps a portion of the cost for itself (e.g, 30%) and gives the rest to the *publisher* of the application.

2.1.3 Ad Networks

The *Ad Network* acts as an intermediary between the *brands* and the *publishers* and provides multiple distribution channels to target the consumer. The *Ad Network* stores the advertisement in its distribution servers and whenever the mobile application requests an advertisement, the *Ad Network* delivers the advertisement.

Currently, there are multiple mobile *Ad Networks* available to a *publisher*, which means that the market is very fragmented and no *Ad Network* is dominant. This is partly due to the fact that each mobile operating system is linked to its own *Ad Network*. For example, Android uses AdMob, while iOS uses iAd. Next, we give a short description of some of the most common *Ad Networks* on the market.

AdMob is an advertising network owned by Google available for Android and iOS platforms. It delivers advertisements to mobile webpages and mobile applications. It charges based on CPC, CPM pricing model, or both and is one of the advertising networks with the highest publisher base [8].

iAd is an advertising network created by Apple, and it is adopted only on the iOS platform (for iPhone/iPad devices). It is the biggest advertising network for the iPhone and iPad devices and uses either CPC or CPM pricing model [8].

Millenia Media[3] is an independent network, its main focus is a bit different, because its primary goal is to create positive brand awareness. Millenia Media publishers are among the Nielsen top 100 sites, meaning that they have a quality publisher base and mainly uses CPM pricing model [8].

InMobi[4] and MadHouse[5] are mobile advertising networks for the Asian market. InMobi is developed and widely used in India, while MadHouse is prevalent in the Chinese market. Similar to the previous advertising networks, both of these advertisement networks implement advertisements for mobile webpages and mobile applications, however both prefer mobile webpages and the CPC pricing model [8].

[3]http://www.millennialmedia.com
[4]http://www.inmobi.com
[5]http://www.madhouse.cn/cn/index.php?sid=

2.2 Advertisement Delivery Process

The *Ad Network* plays a central role in delivering advertisements, it uses the following steps:

1. A *brand* creates an advertisement (such as, text, image, video) and registers it with the *Ad Network*. It also sets a daily budget and the preferred pricing model (CPC, CPM, CPA).

2. A *publisher* integrates the *Ad Network*'s Software Development Kit (SDK) to enable advertisements within its application. The *publisher* also chooses the type of advertisement (typically, banner or interstitials) to display in the application, and has no control over the content of the advertisement. However, the actual content of the advertisement is controlled by the *Ad Network*.

3. The SDK specifies the area and the size of the advertisement, maximum and minimum interval for changing the advertisements (*refresh interval*) and gathers information about the consumer to provide relevant advertisements (*targeting*).

   Targeted advertising is a type of advertising where advertisements are sent based on various traits specific for the consumers [38]. In particular, *Ad Networks* send advertisements based on demographics (location of the user), psyhogaphics (personality, values, attitudes, interests, and lifestyles), behaviour habits (purchase history) or other personal information that *Ad Networks* use to reach as many consumers as efficiently as possible. Currently, *Ad Networks* receives this information with each advertisement request as part of the HTTP message.

4. Next, the mobile application picks a refresh interval to routinely fetch advertisements from the *Ad Network*. If the consumer interacts with the displayed advertisement, the SDK collects the appropriate statistics and updates the analytics.

Figure 2.6 shows the basic interaction between a mobile device and an *Ad Network*. The mobile application integrates the advertisement container, which requests advertisements from the *Ad Network*. The *Ad Network* contains multiple servers to balance the load across multiple geographical locations, so that it is able to target the consumer in the best possible way. The *Ad Network* also implements additional servers for collecting analytics about the advertisements it serves.

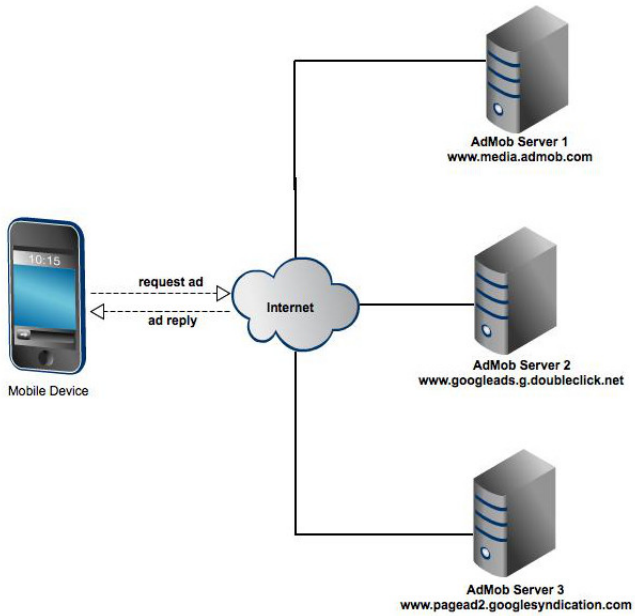


Figure 2.6: Interaction between a mobile device and an *Ad Network* (AdMob).

The *Ad Network*'s SDK provides simple functions and methods that hide the protocol and communication complexity from the developer. The SDK also provides guidelines for better advertisement engagement, they are:

- **How advertisements are embedded?**: Most common form of advertisements inside mobile applications is the banner advertisement and *interstitials*. Both advertisements are presented with an image and text. Banner advertisements appear at the top or bottom of the

screen. However, the interstitials show up as full screen advertisements when the user opens the application.

- **Delivery method**: The advertisements are delivered over Hypertext Transfer Protocol (HTTP), using a Representational State Transfer (REST) Application Programming Interface (API). Figure 2.7 shows an example HTTP dialog between a mobile application and an advertisement server. One can notice, that multiple HTTP requests/replies are used for fetching an advertisement. Also, after each advertisement request there is a time period of no activity, due to the refresh interval. This refresh interval can cause costly transitions between the different radio power modes as a result of staying in high radio state even after the communication is over.

- **Refresh rate/interval**: Advertisements are fetched routinely by the application at a defined interval, which indicates the time period between each advertisement request. This interval is normally set by the publishers. For AdMob, the minimum refresh interval is 30s and the maximum is 120s.

2.3 Challenges of Mobile Advertising

The main concerns of mobile advertising are privacy and security-related issues [11, 25]. These issues have been studied extensively and mainly result from inappropriate advertising practices such as, exposing private informations. One concrete example is that free applications on the Android market use targeted and personalised advertising which takes advantage of user's private information (location, user's contacts, messages) to send advertising messages. If this private information is not transferred using a secure protocol (i.e., over HTTPS) then intermediate nodes can also see this information. Also, if the databases that stores the private information are not adequately protected, the data can be compromised.

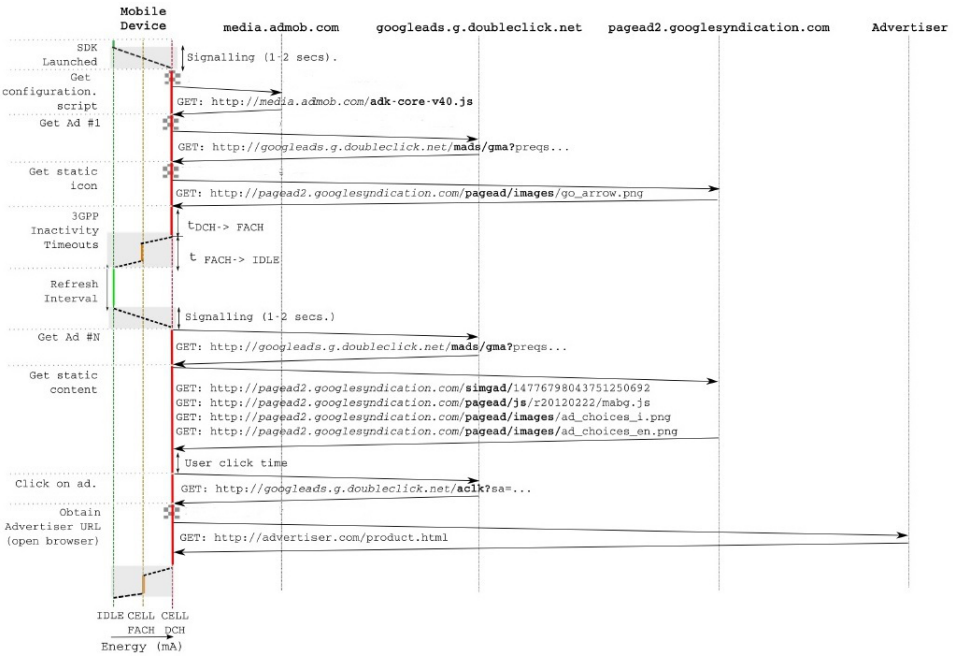


Figure 2.7: HTTP communication between a mobile device and *Ad Network*'s servers for advertisement delivery. Modified from [37]

.

A number of studies [5, 9, 11, 17] have found that advertising has privacy problems. In AdRisk [11] the authors found that most of the free applications on the Android market use 3^{rd}-party advertising libraries, which upload sensitive information aggressively (for example, call logs, address book and browser bookmarks) to the advertising servers and run *unsafe* source code. Similarly, authors of [17] noticed that some free applications on the market request 2-3 times more permissions than paid ones, including user's location, contacts, and messages. [17] proposes to solve the problem by using a privacy-protection mechanism based on feedback control loop that adjusts the privacy level in response to advertisement generated revenue.

MobiAd [14] and PrivAd [13] on the other hand, suggest local profiling of advertisements. The mobile phone gathers and stores user's interests so it can download advertisements that are only relevant to the user. The advertisements are downloaded from a third party proxy which also takes care of user's clicks (to maintain anonymity of the user's preferences). These

proposed solutions involve CPU-intensive cryptographic computations, which on the mobile device directly affects the battery life. Also the communication overhead increases as result of many network operations.

Chapter 3

Energy Implications of Mobile Advertising

In this chapter, we provide background information on energy concerns for mobile devices, starting from battery limitations to identifying applications and components that drain the battery. Additionally, we measure the energy consumption of downloading, and rendering advertisements.

3.1 Background

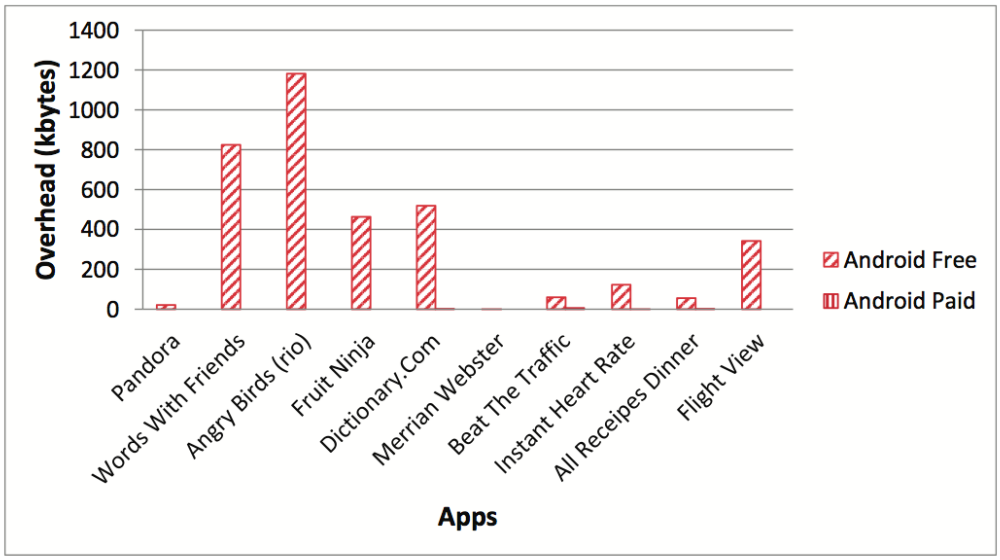
Nowadays, mobile devices are like small computers, i.e., they are computationally powerful, have always-on Internet, more storage and sensing capabilities. These features encourages developers to create more engaging applications but, they also consume more power and hence, limit the mobile phone battery to few hours of operation. Unfortunately, in recent years, the capacity of batteries has not evolved at the same rate as the other physical components. Therefore, the only way to prolong the battery life is to build energy-efficient applications and services. [36] suggests that the only way to increase battery life is to implement energy-efficient mobile OS/platform, smarter and less energy consuming resource management techniques and energy-friendly applications.

One of the major energy consuming components in a mobile phone is the

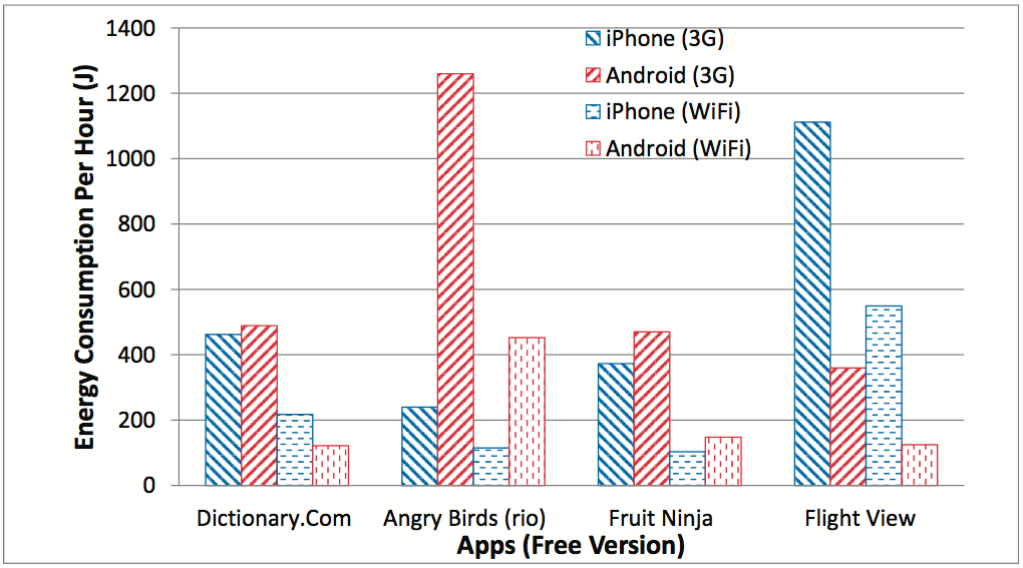
network interface, i.e., transfering data over the network [2, 23, 26]. Hence, mobile applications should consider using the network resources efficiently. Another important consideration is the difference in energy consumption for data transfer over WiFi and 3G networks. The main reason for the difference in the energy consumption is the *3G tail-energy*, which appears because the 3G network interface does not go to sleep and remains in higher power state for several seconds after completing a data transfer [28]. To overcome the 3G's high-tail energy consumption, [2] proposes to schedule data transfers over low cost networks (e.g., GPRS and WiFi) instead of the 3G interface, to minimise the overall time spent in high energy states when transferring data.

Not just the network usage needs to be optimised but optimising at the application-level also results in preserving battery. [35] shows that many web pages and components (CSS, JavaScript, images) are rendered in an inefficient manner which leads to an increase in battery consumption. [23, 24] profiles applications and identifies the most energy consuming components of an application. Their analysis of popular mobile games shows that 65-75% of the total application energy is spent on advertisements, 20-30% on user's location (due to analytics), whereas, only 10-30% is spent on running the core of the application.

[6, 17] found that the mobile advertising ecosystem is mainly based on free mobile applications that use targeted advertising to generate revenues. Because mobile applications transfer the advertisements in small and frequent data transfers theres is an increase in the consumed energy. [39] found that mobile applications that use advertisements seem to have higher energy consumption than mobile applications who do not. [39] have tested ten applications on the Android Market and found that free applications seem to generate more data overhead (advertisements + analytics) than paid applications (see Figure 3.1(a)). They describe the overhead traffic as a combination of *advertisements* and *analytics*. While the advertisement traffic is related to text, images or graphic content that is displayed when the application is used, the analytics traffic refers to the smartphone data that is sent to third party servers to analyse user's behaviour. Beside that, [39] found that (see Fig-



(a) Overhead data of free and paid applications on Android.



(b) Extra energy consumption caused by overhead traffic on iPhone and Android.

Figure 3.1: Taken from [39].

ure 3.1 (b)) the overhead traffic generates as additional energy consumption on the mobile device.

[16] and [2] propose a framework (CAMEO) that aims to preserve the de-

livery and display of mobile advertisements, while dramatically lowering the advertisement related consumption of metered wireless bandwidth. CAMEO acts as an intermediary solution between the mobile application and the *Ad Network* and it uses caching and prefetching mechanisms. To ensure that only the most relevant advertisements are requested, CAMEO uses a prediction component for guessing the user's context. Unfortunately, [16] does not provide implementation and evaluation of its framework.

[37] analysed a day of advertising traffic containing 3 million subscribers. Their analysis involve comparing advertising and aggregate traffic from the users, obtaining frequency of advertisements and average number of requests required for fetching an advertisement. Also, they measure the energy consumption of delivering advertisements to a simple bogus mobile application. Their results show that the advertising traffic is a significant fraction of the total traffic of mobile users and that it increases the energy consumption of the mobile device.

[37] also proposes a solution for reducing the energy consumed by advertisements (fetching and rendering) in mobile applications. Their solution employs caching and pre-fetching custom made advertisements from their proprietary advertising server. However, their implementation does not work with existing advertising networks. A more suitable implementation would be one that reduces energy consumption and integrates well with existing *Ad Networks* (e.g., AdMob, iAd).

3.2 Evaluating Power Consumption of Advertisements

In this thesis, we aim to reduce the energy consumption in a backwards compatible way. While the design of our proposal is similar to the proposed by [37], it differs in implementation and therefore, the energy consumed will be different. To provide a solid foundation to our proposal, we start by establishing the performance of mobile applications by 1) evaluating 5 stand-alone games that use the network only to fetch advertisements, and 2)

varying the advertisement interval of a custom application.

It is important to note that, in this section, we use the term *power* instead of *energy* when we specifically discuss results.

3.2.1 Methodology

We run our experiments on Samsung Galaxy[1] mobile device with Android OS v3.2. For the measurements, all background applications on the device are turned off and no other Internet traffic is present, except the one generated from the advertisements. The only services running on the device are the Android stock services, which uses 85MB from the available 300 MB memory. To measure the energy consumption, we use a hardware tool, Power Monitor[20]. For more details on energy measurements with Power Monitor refer to Section 5.1.

It is important to note that, the results in this chapter are published at [27].

3.2.2 Energy Consumption of Mobile Games

We evaluate the power consumption of 5 games: *Angry Birds* [29], *Fragger* [18], *Yoo Ninja* [32], *Skater Boy* [31] and *Ceramic* [30]. The games are published on the Android Market and use AdMob as the main *Ad Network*. These games do not need to use the network interface actively except for fetching advertisements. We measure the $\Delta PowerUsage$ by comparing the power usage with and without advertisements and represents the power consumed by the application to fetch advertisements.

Table 3.1 shows the power usage of the advertisements in the games. We observe that the $\Delta PowerUsage$ increases for applications which fetch more advertisements. For instance, *Angry Birds* (15%) and *Fragger* (15%), utilise more power than *Yoo Ninja* (8%), *Skater Boy* (4%) and *Ceramic* (4%), because the former set fetches more advertisements compared to the latter.

We also observe that in some cases the AdMob server frequently responds

[1] $http://www.samsung.com/global/microsite/galaxys/index_2.html$

Game	ΔPowerUsage [in mW] (Min–Max)	(Average)	% PowerUsage by Ads	Total no. of Ads Served	No. of Ad Requests denied by Ad Server
Angry Birds	110–136	(131)	15%	67–80	21–30
Fragger	157–225	(157)	15%	32–39	12–14
Yoo Ninja	70–158	(71)	8%	14–17	2–7
Skater Boy	40–53	(42)	4%	6–8	0–1
Ceramic	66–88	(43)	4%	6–7	0

Table 3.1: Average power consumption of advertisements in mobile games over WiFi. The gameplay duration is 15 minutes.

to advertisement requests with an HTTP 304 (Resource Not Changed) Response. When an application requests for advertisements with shorter refresh intervals then allowed (e.g., every 10s instead of the minimum 12s), the *Ad Network* responds with HTTP 304 Response, which means that the advertisement content has not changed and new advertisement is not sent back. We speculate that this happens when a user completes or resets a game level quickly. The following are fraction of the HTTP requests that result in no new advertisements: *Angry Birds* (37%), *Fragger* (35%), and *Yoo Ninja* (11%). Figure 3.2 shows all the advertisement requests over time for *Fragger*, as well as the successful (HTTP 200) and unsuccessful (HTTP 304) responses.

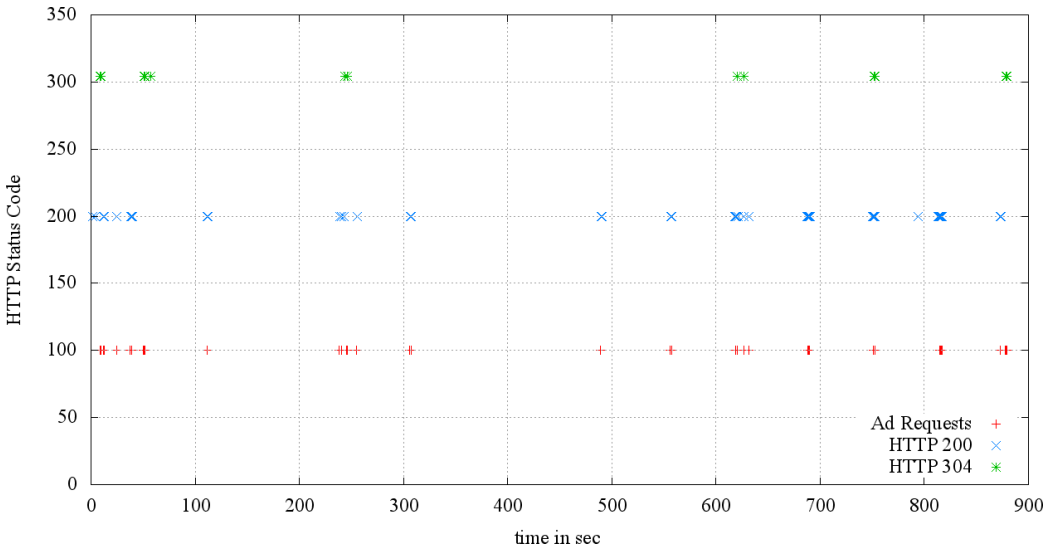


Figure 3.2: Advertisement Server responds with a HTTP 304 (*Not Changed*) Response.

In Table 3.1, we compare fetching advertisements over 3G with WiFi. We

Game	Ads over WiFi	Ads over 3G
Angry Birds	15%	27.7%
Yoo Ninja	8%	18.4%
Ceramic	4%	9.6%

Table 3.2: Comparison of average power consumption of advertisements over 3G and WiFi network

.

observe that in some cases the power consumption for fetching advertisements over 3G reaches up to 27% of the total application's energy consumption. This increase in energy consumption is due to: a) *capacity dissimilarity*, the 3G bandwidth is lower than WiFi, hence, downloading content takes more time, attributing to higher power usage ; b) *tail-energy*, the 3G radio stays in the high energy state for a longer duration before returning to the idle state [28], this is particularly harmful when transferring small amounts of data [2, 7].

3.2.3 Impact of Advertising Interval

We built a custom application to just fetch advertisements from the AdMob servers. The application contains an advertisement slot at the bottom of the screen, and no other components to avoid any additional CPU, network, or memory overhead, which might affect the energy measurements. We mainly use this methodology to accurately measure the energy cost of fetching and displaying advertisement in the application. The advertisement slot size used in all experiments is 480x800 pixels. Advertisements are requested with specific refresh intervals (12-120 seconds) over WiFi and 3G networks. The 3G connection is a 1 Mbit/s connectivity from *Elisa*, whereas the WiFi is connected to the Aalto university's infrastructure.

Table 3.3 shows the average power consumed by the application to request and display advertisements. We observe that for short refresh intervals, the power consumed by the application is higher, while for long intervals it is lower. In particular, when advertisements are fetched with a 30s refresh

Advertisement Frequency	**ΔPowerUsage** [in mW] (*Average*)	**% PowerUsage by Ads**	**Total No. of Ad Req. in 15 mins**	**No. of Ad Requests denied by Ad Server**
every 12s	97	21%	75	17
every 15s	90	19%	60	9
every 30s	82	18%	30	5
every 45s	80	17%	20	0
every 60s	67	15%	15	0
every 90s	52	12%	10	0
every 120s	56	13%	7	0

Table 3.3: Average power consumption of advertisements in a baseline application running for 15 minutes. Advertisements are downloaded over 3G network. The baseline power consumption for an application without wireless and advertisements is about 370 mW. The total number of Ad Requests include the requests that were denied by the AdMob Server.

interval, the power consumed is 82 mW, and when it is fetched with a 90s interval, it is 52 mW.

3.2.4 Summary

To summarise, the current approach of delivering advertisements at fixed intervals over a 3G network can be up to a quarter (27.7%) of the applications' total power consumption. We also observe that the *Ad Network* occasionally responds with a HTTP 304 (Resource Not Changed) when requesting advertisements, which suggests that the application is needlessly requesting for advertisements without actually receiving new ones.

Chapter 4

Energy Aware Solution for Mobile Advertising

The current mobile ecosystem encourages publishers to use advertisements in their mobile applications for the purpose of increasing their advertising revenues. Many applications that do not require Internet connection as part of the application, are turning to online applications where the Internet connection is necessary. In recent years, the advertisement traffic volume increased significantly, nearly 1% of mobile users downloaded 2MB of advertisement traffic data per day [37]. In Section 3.2.3 we show that the energy consumption of receiving advertisements in mobile games can reach up to a quarter of the total power consumption of the app.

In this chapter we propose an Energy Efficient Ad Network (*eeAdNetwork*) that aims to reduce energy consumption for delivering advertisements over the Internet. *eeAdNetwork* is compatible with existing *Ad Networks* (i.e., AdMob for Android platform). In the following sections, we discuss the *eeAdNetwork*'s design and implementation in more details.

4.1 eeAdNetwork Design

eeAdNetwork is a middleware that aims to reduce the energy consumption of delivering advertisements by 1) pre-fetching multiple advertisements in

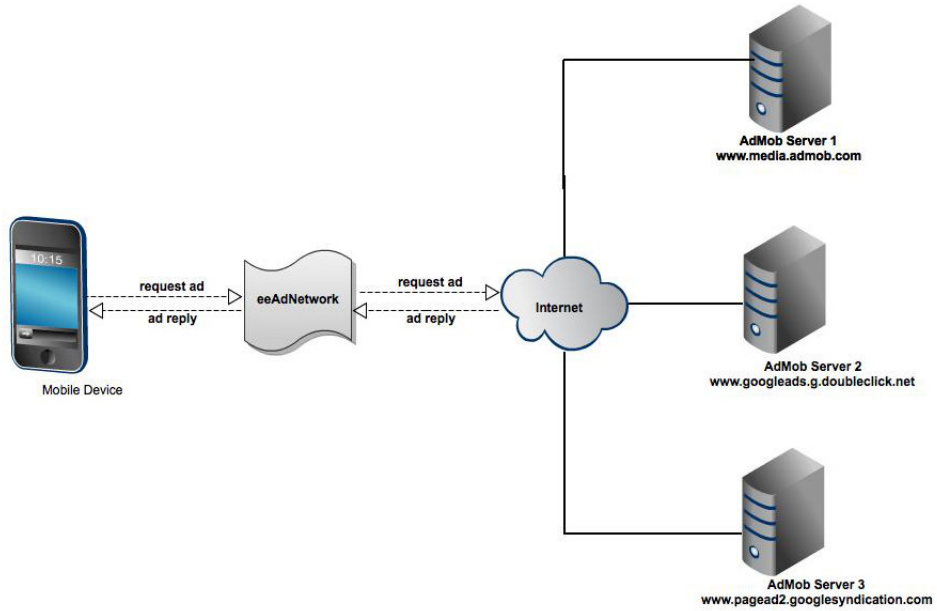


Figure 4.1: eeAdNetwork as a intermediary solution between the mobile device and the *Ad Network* servers.

a single request and storing them in a local cache, and 2) serving appropriate (context relevant) advertisements from the local cache. Technically, *eeAdNetwork* works between the mobile application and *Ad Network*. Figure 4.1 shows the *eeAdNetwork* positioned between the mobile device and the *Ad Network*. Whenever a mobile application requests an advertisement, *eeAdNetwork* intercepts the requests, and instead of fetching the advertisement provides one from the local cache to the application. The *eeAdNetwork* prefetches the advertisements from the *Ad Network*.

This proposed solution aims to solve three problems; First, reduce the number of advertisement requests to a minimum without actually decreasing the number of advertisements shown to the user. Second, generate an appropriate advertisement from the local cache. Lastly, fill up the cache with advertisements when WiFi (or GPRS instead of 3G/LTE) is enabled, or when the network interface is actively used by another application.

To enable this, at first the *eeAdNetwork* intercepts all the HTTP requests from the application, hence, the requests are received and interpreted

by *eeAdNetwork* on the mobile device, instead of the *Ad Network*. Once *eeAdNetwork* identifies an advertisement request, it responds with an advertisement from the local cache (on the mobile device).

The *eeAdNetwork* can fill-up the cache with advertisements in multiple ways: a) *passively*, b) *actively*, and c) *opportunistically*. To *passively* fill the cache, the *eeAdNetwork* intercepts the advertisement requests but passes them through, instead it intercepts the responses (containing advertisements) and stores them in the cache. To clarify, the cache will store advertisements only when the mobile application asks for an advertisement. Once the cache is full, it intercepts the advertisement requests and sends the application an advertisement from the local cache. Thus, in the *passive* case, advertisements are fetched individually by the mobile application. This is particularly useful, if the *Ad Network* does not allow fetching the advertisements in a burst.

To *actively* fill-up the cache, advertisements are downloaded concurrently, usually when triggered by some action (i.e., the cache needs to be refreshed, or cache is empty, or user-context changes). In this case, the advertisement are fetched in a burst instead of one by one and multiple advertisements are received in one network transfer.

To *opportunistically* fill-up the cache, the *eeAdNetwork* monitors the activity on the network interfaces. If the network interface is being actively used by another application, the *eeAdNetwork* uses this occasion to fetch advertisements from the *Ad Network* and this does not cause any penalty on the energy savings because another application was already using the network interface.

The advertisements stored in the cache are replayed multiple times. To guarantee their freshness a Time To Live Value (TTL) is assigned to each advertisement. When the TTL expires, the advertisement is removed from the cache and new advertisement is requested from the *Ad Network*. If the user-context changes (new geographical location), the cache can be flushed and refreshed immediately by the *eeAdNetwork*. In this case the current advertisement are deleted and new ones are downloaded.

User's context is personal information gathered by the *Ad Network*, typically, the location, the types of applications installed, age-range, male or fe-

male [16]. *Ad Networks* use this information to send targeted advertisements or advertisements that are relevant to the current user. While location has been demonstrated to affect the displayed advertisements for web applications [12], the range of context used for mobile devices is potentially much larger. Therefore, *eeAdNetwork* should support targeted advertisements.

One way for *eeAdNetwork* to support targeted advertisements is by learning user's context over time and predicting it. Another way is to track user's context changes and whenever there is a change in context information, the *eeAdNetwork* flushes the cache and fetches new advertisements. However, if the user's context changes too often (user is driving), *eeAdNetwork* can choose between two options: 1) caching - show the current advertisements from the cache beside their irrelevance for the current location or 2) disable caching - the application uses the standard approach of delivering advertisements relevant to user's context. The main trade-off is between both options is the advertisement relevancy and energy savings.

4.2 eeAdNetwork Implementation

The design discusses the high-level concepts and features involved in *eeAdNetwork*. In this section, we discuss the implementation details of *eeAdNetwork*. Figure 4.2 shows the complete *eeAdNetwork* system. It consists of two parts: a) a *mobile part* and b) a *server part*.

The *mobile part* consists of the following components:

- *Advertisement Selector* component responsible for choosing advertisements from the cache and sending it back to the requesting application.
- *Local Cache* component that stores advertisements on the filesystem of the mobile phone.
- *Advertisement Optimiser* component that maintains and monitors the *Local cache*. *Advertisement Optimiser* component is able to flush the cache and fetch new advertisements in collaboration with the *Ad Pre-Fetch server*.

Beside these components, the mobile client also runs a socket server as a background service. The service listens on a port for advertisement requests.

The *server part* mainly consists of an *Ad Pre-Fetch server* that acts as a gateway between the mobile device and the *Ad Network* servers. It is also capable of storing and prefetching advertisements from the *Ad Network*. Additionally, the *mobile part* communicates with *server part* to fetch advertisements in a burst because this feature is not yet implemented by the *Ad Networks*.

Ads requesting application (Ads app) is any mobile application that requests advertisements. For our experiments, we built a custom application that requests advertisements at specific refresh intervals. The advertisement refresh interval is configurable and adjusted in each of our experiment.

4.2.1 Interception

When the *Ads app* requests for an advertisement, the background service of the *eeAdNetwork* intercepts and redirect the request to its local cache.

One way to implement this functionality is using a Web proxy that simply inspects the target domain names (of the Ad Network) in the HTTP requests. If the domain names involve any of the AdMob servers then it redirects the message to the mobile phone instead of the AdMob servers. Implementing this technique is fairly simple, however, there are implications on energy consumption. A Web proxy consumes high CPU & network resources, therefore, we use a simpler and more energy-efficient approach.

Rather than intercepting HTTP messages, we intercept DNS queries and redirect them to our background application that handles them. We perform the DNS redirects by modifying the host names mappings in the host names file. On linux-based platforms this can be achieved easily by modifying the `/etc/hosts` file, but for other platforms (e.g., Windows Mobile) using a Web proxy might be easier choice. The details of the name mappings for `/etc/hosts` file is explained in Section 5.1.

4.2.2 Local Cache

Once the advertisement requests are intercepted and redirected, if the local cache contains advertisements, the *eeAdNetwork* provides an appropriate advertisement from the cache.

The cache is implemented as a *hash map* data structure stored in a file on the persistent storage (filesystem) of the mobile phone. To prevent modification or deletion by the mobile phone owner, the file is assigned with special access rights (permissions). The file ownership and actions are only available to the *eeAdNetwork* service and no other user. This way, the file can not be viewed, modified or deleted by other users of the mobile device.

In *eeAdNetwork*, we implemented two approaches to fill-up the cache with advertisements: *passive* and *active*. In the first case, the *eeAdNetwork* intercepts advertisement requests made by the *ads app* and stores the advertisements returned by the *Ad Network* into its cache. Therefore, the *eeAdNetwork* only relays the advertisement requests to the *Ad Network*, but it stores the HTTP responses (containing the actual advertisement). Finally when the cache is full, the *Advertisement Selector* component picks an advertisement from the cache and sends it to the *ads app*.

In the *active* case, *Ad Optimiser* component is used to pre-load the cache. Since the AdMob platform does not allow fetching advertisements in a burst, we implemented this feature as an add-on in our *Ad Pre-Fetch Server*. Currently, the AdMob platform only allows fetching advertisements at a fixed interval, therefore, the *Ad Pre-fetch Server*, works on behalf of the user and pre-fetches and stores the advertisements in the *Ad Pre-fetch Server* cache. The *Ad Optimiser* collaborates with the *Ad Pre-Fetch server*, to fetch these advertisements in a burst. To summarise, the *Ad Pre-Fetch server* downloads and stores the advertisements using a fixed interval. And when requested by the *Ad Optimiser*, it returns a group of advertisements in a burst.

The TTL value of the advertisements is represented with a numeric value that is decreased every time an advertisements is shown to the user. When TTL reaches zero, the cache replaces that specific advertisement with a new one.

4.2.3 Analytics and Targeting

The *Ad Optimiser* is capable of flushing the cache and requesting a new set of advertisements from the *Ad Pre-Fetch server*. However, our system is not yet deployed and is a proof of concept. Therefore, the advertisement targeting is a work in progress that we discuss in Chapter 6.3.1.

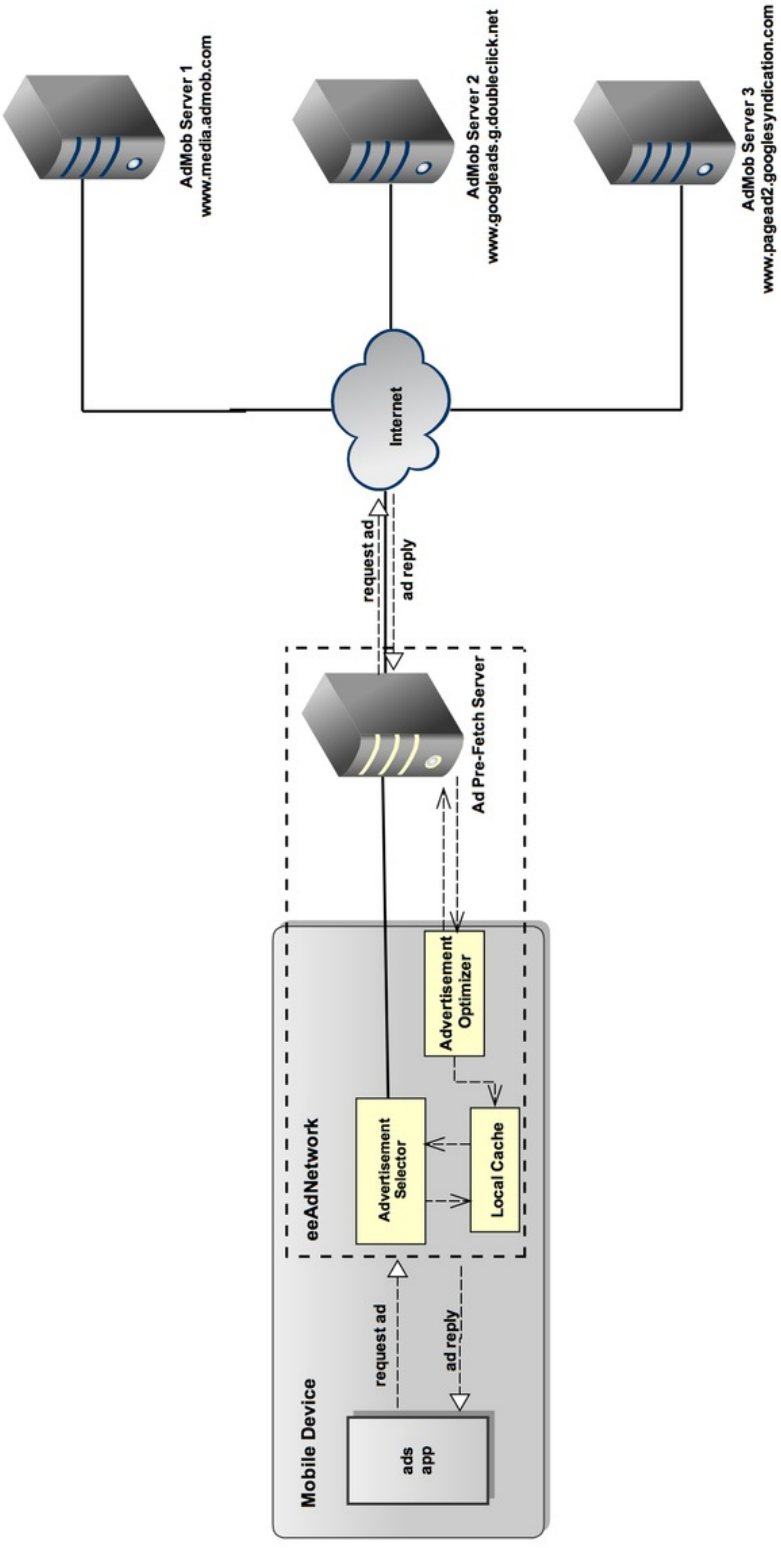


Figure 4.2: Integration of eeAdNetwork and eeAdNetwork components.

Chapter 5

Performance Evaluation

In this chapter, we evaluate the performance of our proposal, *eeAdNetwork* and compare it with the standard approach. To measure the performance in a controlled environment, we build a custom-made mobile application that requests advertisements at specific intervals, we measure the power consumption of this application when *eeAdNetwork* is enabled and disabled. Before discussing the results, we present the methodology and the measurement environment (hardware and software setup, power measurement tool).

5.1 Measurement Environment

All the experiments are performed on a Samsung Nexus S (developer's version) [33]. The technical specifications of the mobile device and the *eeAdNetwork Ad Pre-Fetch Server* can be found in Table 5.1. Nexus S runs Android 4.1.2 Jelly Bean operating system.

Figure 5.1 shows the network topology, the smartphone runs the custom-

	Nexus S	**MacBook Pro**
OS	Android 4.1.2 Jelly Bean	OS X Mountain Lion 10.8
Processor	1 GHz Cortex-A8	Intel Core i5 2.5 GHz
RAM	512 MB	8GB of 1600MHz DDR3L
WiFi	802.11 b/g/n	802.11 a/b/g/n

Table 5.1: Devices used in the experiments.

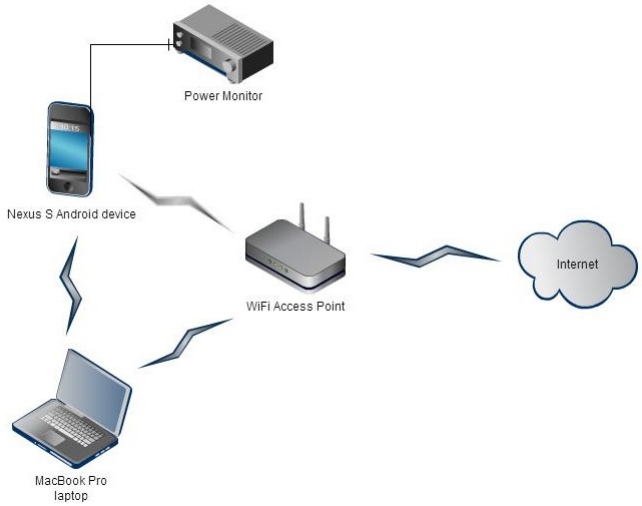


Figure 5.1: A network topology of the devices.

made mobile application. The *Ad Pre-Fetch Server* that relays advertisements requests/replies enables refreshing of the phone's cache at once is running on the laptop machine. The mobile device uses WiFi or 3G network for fetching advertisement. When WiFi is enabled on the device, cellular data transfers are disabled, and, vice versa, to avoid the OS from automatically switching between the radio interfaces. We use the Power Monitor [20] to measure the power consumption of the device.

Android operating system is based on the Linux kernel, therefore access to some low level services on the device requires administrator permissions[1]. The main purpose for rooting the device is to allow administrator permission for intercepting the advertisements by configuring IPTABLES[2] and modifying `/etc/hosts`.

To intercept the advertisement requests, we do the following: 1) implement a server socket on the mobile device that listens for traffic and 2) we redirect all advertisement requests for the AdMob servers to the server socket by resolving the DNS queries to it. We map the the AdMob's hostname to our server socket in `hosts.txt` file. For example, we map `http:`

[1]To root the Samsung Nexus S, follow the procedure documented in [21].
[2]http://www.netfilter.org/projects/iptables/

localhost	127.0.0.1
127.0.0.1	media.admob.com
127.0.0.1	googleads.gdoubleclick.net
127.0.0.1	pagead2.googlesyndication.com
127.0.0.1	csi.gstatic.com

Table 5.2: Contents of `/etc/hosts` file on the smartphone.

`//media.admob.com` to `127.0.0.1`, and hence, intercept and forward all advertisement requests for the AdMob servers to our server socket. The content of the `/etc/hosts` file is shown in Table 5.2.

Additionally, the requests intercepted by `/etc/hosts` file is by default sent to `port 80`. We use IPTABLES to forward these requests to our server socket listening on a specific port (`8080`) [22]:

```
$iptables -t nat -A OUTPUT -p tcp --dport 80 --to-destination:127.0.0.1:8080
```

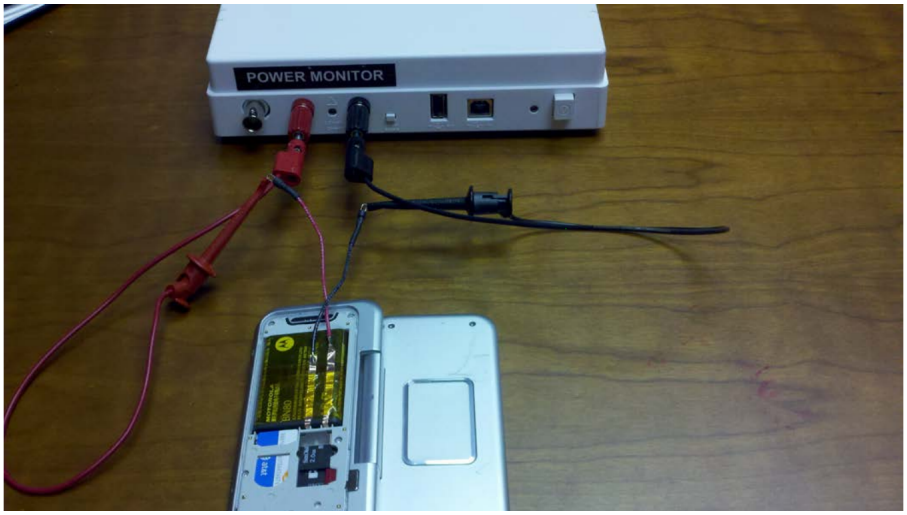


Figure 5.2: Power Monitor hardware device connected to a mobile phone to measure energy consumption. Taken from [20]

We use the Power Monitor [20] hardware device to measure the mobile phone's energy consumption. The Power Monitor is a robust and efficient tool that can measure power (in mW), current (in mA), voltage (in V), and consumed energy (in uAh) for any mobile device. The Power Monitor measures power by directly connecting to the mobile device's battery com-

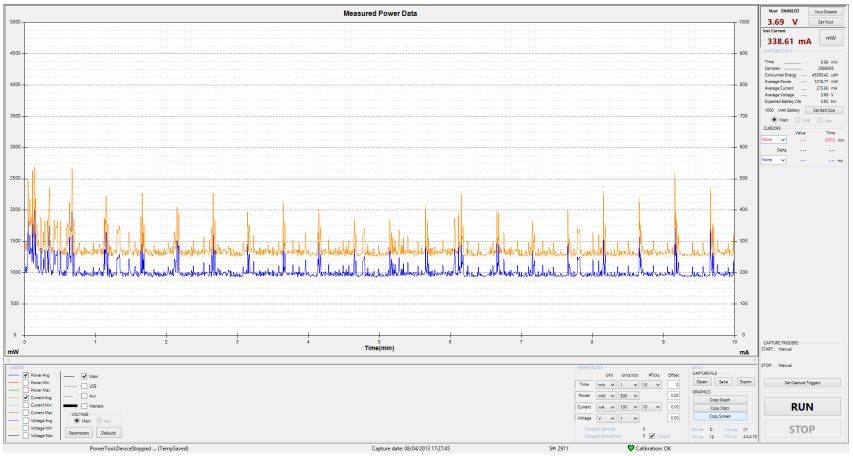


Figure 5.3: Power Tool user interface displaying current energy details (power and current).

partment with copper wires (see Figure 5.2). In particular, the battery of the the mobile phone is bypassed, and the power terminals of the mobile device are directly connected to the power supply of the Power Monitor. Consequently, the phone is powered by the Power Monitor instead of its own battery. Power Monitor measures energy (all related metrics) at a resolution of 200 microseconds.

The Power Monitor hardware is complemented with measurement software that shows the measured power with graphs and tabular data. Figure 5.3 shows an example of power (blue line) and current (red line) consumption in mW and mA, respectively. In addition to the graphical representation of the measured data, Power Tool software can also export the data in comma-separated values (CSV) files that can be later used for analysis.

5.2 Methodology

We implement a custom-made application that fetches advertisements at fixed intervals. To evaluate the power consumption of *eeAdNetwork* we measure the power consumption of the custom-made application when 1) *eeAdNetwork* is enabled (*caching on*), and 2) *eeAdNetwork* is disabled (caching

off). We use 30, 60 and 90s refresh intervals and run each experiment for 1 hour. Additionally, we run each experiment over WiFi and 3G to measure the impact of these access technologies on the energy consumption.

It is important to note that our main goal is not to change the current way of showing advertisements. We do not build new *Ad Network*, but we propose and implement a backwards compatible solution that works with existing *Ad Networks* (e.g., AdMob on Android) and that aims to make advertising energy-efficient.

5.3 Choosing the right cache size

In our earlier experiments with mobile games (See Section 3.2.2), we observe that the AdMob Server repeatedly responds with an HTTP 304 (Content Not Changed) when requesting advertisements. Moreover, while running our experiments, we observed that the advertisements were not changing often and the same advertisement was shown multiple times. To verify, we run the custom-made application with varying advertisement refresh interval for 12 hours while the *eeAdNetwork* keeps track of the advertisement requests and responses.

Figure 5.4: An example of an HTTP response header used for discovering unique advertisements.

Our analysis of the logs shows that the *eeAdNetwork* receives 120 adver-

tisement requests per hour, from which only 8 are unique. Therefore, we set the cache size of *eeAdNetwork* to 8 advertisements. At the end of each hour the cache is flushed and 8 new advertisements are requested from the Ad Pre-Fetch Server. During the hour, the advertisements are replayed equal number of times.

An example of detecting unique advertisements is shown in Figure 5.4. We detect the unique advertisements by parsing the HTTP response header and note the `adurl`.

5.4 Results

The results of the experiments show the comparison of the energy consumptions of the custom-made mobile application when the *eeAdNetwork* is disabled (no caching) and when it is enabled (caching is on). Furthermore, we test the performance of *passive* and *active* caching strategies.

5.4.1 Baseline

Table 5.3 shows the baseline power values of Samsung Nexus S in *suspended*, *idle* and *active* modes. The *suspended* and *idle* values refer to the power mode of the device e.g., the *suspended* mode of the phone is when the backlight is switched off (no visible display activity), while *idle* mode is when the backlight is switched on. The *active* mode is when the custom made application is running in the foreground, but in this case is not fetching advertisements. These values for the *active* mode aim to describe the minimum power usage of the mobile application running in the foreground, without any CPU or network activity. We observe that when the mobile device is in *idle* mode the power consumption is higher than in *suspended* mode. This is as result of the high power consumption that the display consumes. Also, we observe that higher display brightness leads to higher power consumption. Similar observations are found in the work of [4, 26]. The values for all modes are very similar for 3G and WiFi network because network connectivity is not used.

Baseline	**Power usage 3G** [mW]	**Power usage WiFi** [mW]
Suspended	26 (22.0)	32 (19.6)
Idle (Min Brigthness)	617 (8.7)	633 (3.9)
Idle (Max Brigthness)	1199 (14.3)	1196 (7.6)
Application in foreground	965	970

Table 5.3: Average baseline values of Samsung Nexus S. In brackets, the standard deviation.

Advertisement Frequency	**Power usage 3G** [mW]	**Power usage WiFi** [mW]
every 30 sec	1204 (52.2)	1029 (7.7)
every 60 sec	1102 (17.1)	1026 (11.4)
every 90 sec	1065 (24.5)	1015 (14.9)
every 120 sec	1061 (14.3)	1029 (39.9)

Table 5.4: Average power consumption of a custom-made application that fetches advertisements at specific intervals. In brackets, the standard deviation.

We measure the energy consumption of the custom-made mobile application when *eeAdNetwork* is disabled (no caching). The application uses the following advertisement refresh intervals: 30s, 60s, 90s and 120s. These intervals determin the number of advertisements downloaded in an hour. For example, using a 30s advertisement refresh interval the application will request 120 advertisements in an hour. While using a 120s advertisement refresh interval the application will request 30 advertisements in the same period. Therefore, the variation in energy consumption of the application is directly related to the advertising interval or the number of advertisement requests. The Table 5.4 shows that the power consumption is higher when requesting more advertisements and lower when requesting fewer advertisements. It also shows that the power consumption for 3G is much higher than in the WiFi case, mainly due to the 3G *tail-energy*. The results in Table 5.4 are different from the one in Section 3.2.3 because they use different devices with different hardware specifications (CPU, memory).

5.4.2 Passive Cache

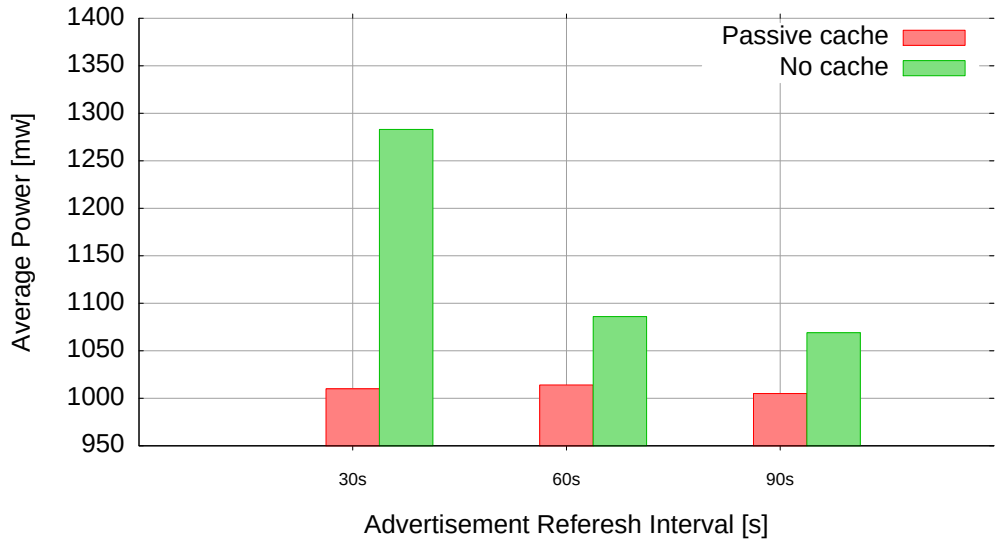
In passive caching the application requests the advertisements independently and the cache relays the requests until 8 unique advertisements are downloaded after which it replays the advertisements from the cache. The time required to download 8 advertisements varies for the different refresh intervals (30, 60 or 90). Figure 5.5 shows the results from passive caching, where we observe that the power consumption of passive caching is lower than no caching.

We also evaluate the performance of the passive cache when using 3G (see Figure 5.5 (a)) and WiFi (see Figure 5.5 (b)). The power savings in the 3G case are much higher because after the cache is filled up, the cache avoids the penalty of 3G tail-energy. While the power savings in the WiFi case are much lower than the 3G case, they are still important because the cache avoids fetching extra advertisements. Therefore, we conclude that *eeAdNetwork* enables higher energy savings for 3G than for WiFi, and in general the passive cache lowers energy consumption.

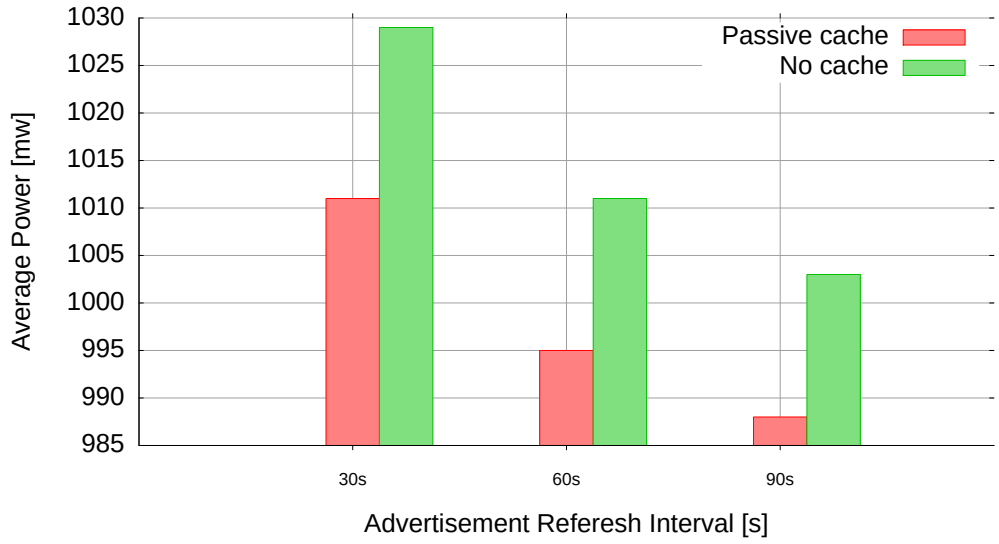
5.4.3 Active Cache

In active caching the cache is flushed and refreshed in the background, therefore advertisements are already pre-loaded in the cache. Whenever an application requests for advertisement, it is replayed from the cache immediately. Figure 5.6 shows the results of active cache experiments. As expected the results show that the energy consumption of the mobile application using active caching is lower than without caching.

As before with passive caching, we observe that the energy consumption of active caching is much lower for 3G, than for WiFi, because in this case the advertisements are fetched in a burst and it bears the penalty of 3G *tail energy* only once.

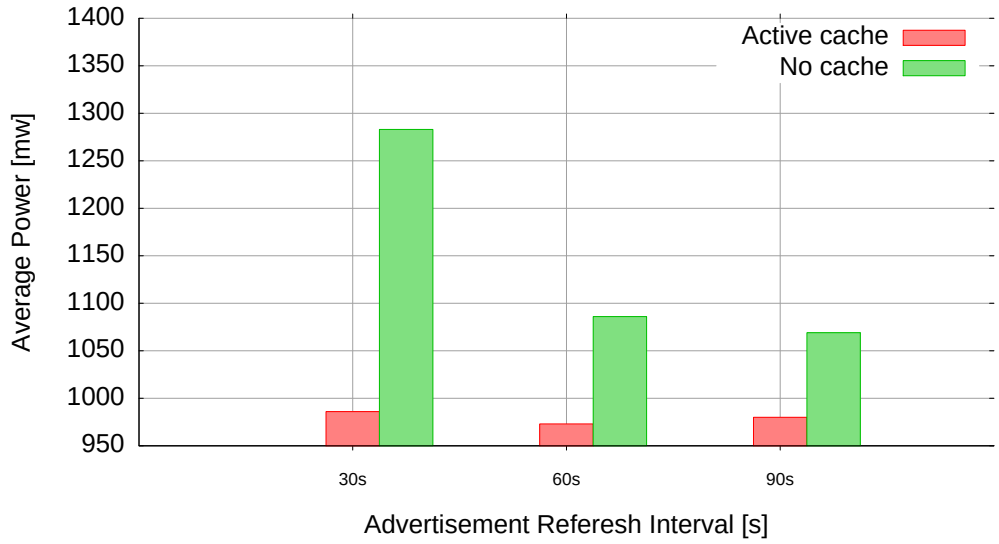


(a) Mobile Network (3G).

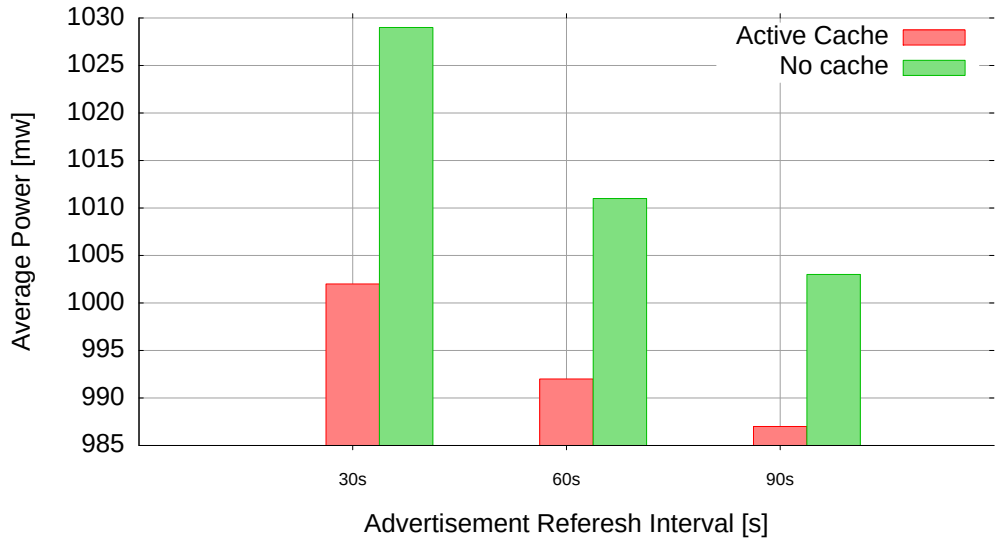


(b) Mobile Network (WiFi).

Figure 5.5: Average power consumption of the custom-made application in passive mode.

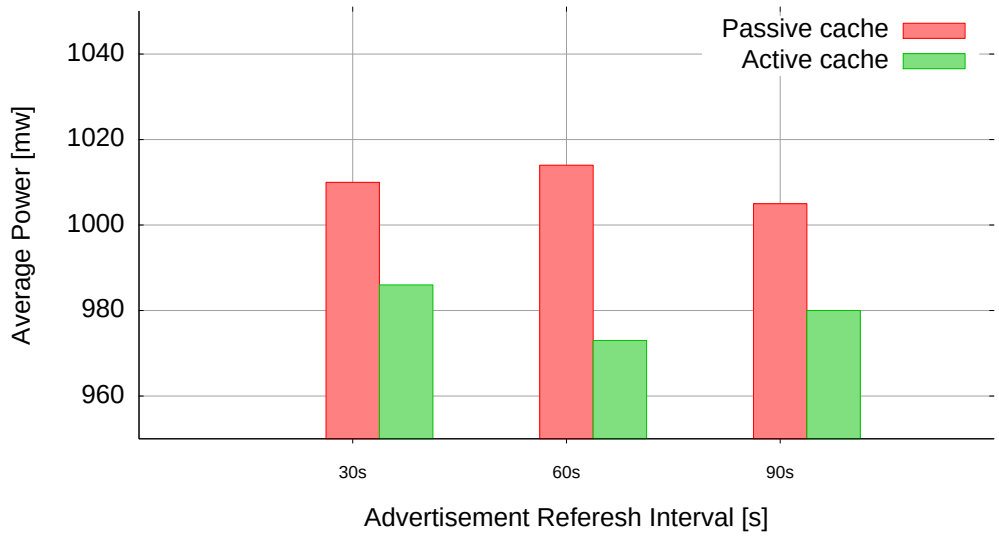


(a) Mobile Network (3G).

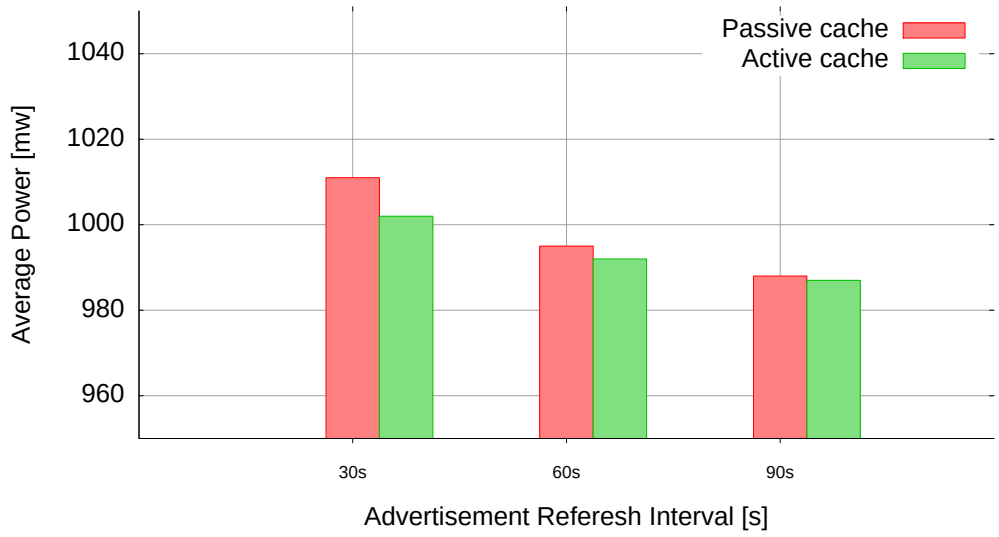


(b) Mobile Network (WiFi).

Figure 5.6: Average power consumption of the custom-made application in active mode.



(a) Mobile Network (3G).



(b) Mobile Network (WiFi).

Figure 5.7: Comparison of power usage of active and passive caching in *eeAdNetwork*.

5.4.4 Passive vs. Active Caching

In active caching, the *eeAdNetwork* fetches the 8 advertisements at once, while in for passive caching it waits for the application to fetch the advertisements. Therefore, the energy consumption in the passive cache is dependent on the advertising refresh interval. Figure 5.7 compares the power consumption of the active and passive caches over WiFi and 3G network access. We clearly note that the power consumption due to active caching does not vary too much across advertisement refresh intervals, while the power consumption due to passive caching decreases with increasing advertisement refresh interval.

To conclude, by using caching the *eeAdNetwork* reduces energy consumption. We observe that the advertisements account for up to 4% of the application power consumption when *eeAdNetwork* is enabled and 25% when it is disabled.

Chapter 6

Discussion

In this chapter, we summarise the results and compare it to results obtained by previous research work. Additionally, we discuss the limitation of our current solution, possible solutions and considerations for deploying *eeAdNetwork*.

6.1 Summing up the Results

From the results obtained in this thesis we make the following conclusions:

- We show that advertisements use up to 27% of application's total energy consumption when fetching advertisements at fixed intervals.
- We implemented our own solution - *eeAdNetwork* that improves the energy consumption and the advertisements only account for 4% of the application's total power consumption.
- We show that AdMob sends only 8 unique advertisements per hour and configure *eeAdNetwork* to cache 8 advertisements per hour.
- Active cache allows higher energy savings than passive cache.

It is important to note that the experiments performed in this thesis cover only the Android and AndMob platform. While our solution can be ported to

other platforms, the energy consumption is expected to be different. Using a different advertising network (not AdMob) may also result in different results, if the advertising network behaves differently.

6.2 Comparison to other published results

Many papers [2, 16, 17] discuss energy concerns of mobile advertising, but do not provide an implementation of their solutions. [37] on the other hand, proposes and implements a caching and pre-fetching solution (called AdCache) that preserves energy in mobile advertising. [37] found that AdCache halves the power consumption of the application while using a 20s refresh interval.

There are key differences between AdCache and our proposal *eeAdNetwork*: First, AdCache is not compatible with existing *Ad Networks*, it proposes its own *Ad Network* with its own mechanisms for delivering advertisements. *eeAdNetwork* on the other hand, is a backwards compatible solution and works with the current *Ad Network*. Also, *eeAdNetwork* uses AdMob's existing functionality for delivering advertisements to the device, but it implements energy-efficient add-ons or mechanisms for fetching and displaying advertisements inside mobile applications. Second, AdCache uses a bogus server for delivering advertisements to the mobile device. However, the authors do not clarify how these advertisements are requested and delivered to the device. The main open issue is: *does the AdCache use single or multiple HTTP requests to fetch the advertisements? are the advertisements concatenated in one HTTP message?* [37]. This information is rather crucial to compare to our design, because if AdCache uses only one server to deliver advertisements and only one request to achieve it, their results differs from AdMob's which requires multiple HTTP messages to download a single advertisement (as observed in Section 2.2).

For this reasons, we can conclude that AdCache and *eeAdNetwork* are actually different architectures based on different implementations and therefore the results are not comparable.

6.3 Limitations

In this section, we list the main limitations of *eeAdNetwork*:

User's context change: the *eeAdNetwork* design is capable of handling user's context change (see Section 6.3.1) but the current implementation ignores the changes in user's context and refreshes the cache every hour. Therefore, if user's context changes (e.g., the location, application, etc.) the *eeAdNetwork* keeps sending the cached advertisements which may not be most appropriate advertisement for that location. While the implementation is easily fixed, the caching strategies require some further study and analysis.

Administrator privileges: The interception and redirection of advertisement messages on the mobile device are implemented using a low level commands and techniques that require root access on the device. This is one limitation that users would face if they are deploying our solution on their devices. However, this concern goes away if *eeAdNetwork* is incorporated into AdMob.

Number of advertisements to cache: *eeAdNetwork* caches 8 advertisements, this valus was picked by running a simple experiment. However, this requires further study, but this value is configurable. Alternatively, this can be configured by the Ad Netowrk (e.g., by AdMob).

6.3.1 Changing User's context

User's context is normally related to personal information that is important to the *Ad Network* for targeting advertisements. Currently, AdMob receives user's context with each advertisement request as part of the HTTP request. AdMob takes this context information and decides if it needs to respond with a new advertisement. For example, AdMob will send an advertisement containing information for a store in a shopping mall, whenever the user is at the shopping mall.

According to [16], served advertisements in mobile application are based on the following context data:

- Mobile network in use (evidence by advertisements suggesting switching

to different carrier).

- Application in use (evidenced by advertisements for other products from the developer).
- Model of the phone in use (evidenced by the advertisements for newer phones).
- Location of the user, on city-level granularity (evidenced by advertisements for an event in the city).

For the purpose of displaying the most relevant advertisements to the users, we have considered the following changes to *eeAdNetwork*:

Mobile network in use: If the mobile network changes (e.g., WiFi to 3G), *eeAdNetwork* will wait a few minutes before flushing and refreshing the cache with new advertisements relevant to the current network. We speculate that the switching between networks does not occur at short timescales, therefore this would not cause any major energy side effects.

Application in use: If the category of the mobile applicaiton in use changes[1], *eeAdNetwork* would update the cache again with relevant advertisements.

Model of the phone: on a new phone, the cache will download advertisements appropriate for it.

Location of the user is the most important context data, because *Ad Networks* use it to send relevant advertisements to the users. *eeAdNetwork* can receive user's location information from two sources: 1) the device and 2) the mobile network. On the device, the GPS sensor or Android's Passive Location Provider[2] provides location information. In addition, the *Ad Network* can query services like, Skyhook[3] to keep track of the user. By using Skyhook, the mobile phone saves one HTTP request. Obtaining location information from the device (GPS & Android's Passive Location Provider) is the most convenient way, however it is also the more energy consuming cite.

[1]the category is typically available in the metadata of the application store

[2]AndroidÕs PassiveLocationProvider reports location changes ob- tained from A-GPS

[3]http://www.skyhookwireless.com

Therefore, the best approach for *eeAdNetwork* is to obtain location information from the GPS sensor or the Android's Passive Location Provider only if thye are active. If either option is unavailable, *eeAdNetwork* would use a service similar to Skyhook for positioning. Once the location information is obtained, *eeAdNetwork* updates the cache. If the user is moving (in a car or bus), their location is changing rapidly, however, this requires further study because the cache does not know the range for which the advertisement is valid. This range information for each advertisement needs to be shared between the cache and the Ad Network.

To keep the cache fresh and filled with relevant advertisements to the user's location and maintain low energy consumption, we need to answer the following questions:

- How quickly does the user move on average: walking, driving.
- What is the location specification required by the *Ad Networks*: city level granularity, region level granularity or latitude and longitude.

6.4 eeAdNetwork in the Real World

So far we have implemented a prototype solution that can be deployed as is, with minor modifications. *eeAdNetwork*'s current implementation can be modified to run on various mobile devices and to support increase in user traffic (scalability).

In the current mobile advertising eco-system *eeAdNetwork* can be deployed by different entities: a) *Ad Networks* b) mobile service operators and c) mobile phone users. *Ad Networks* can integrate the mobile side of *eeAdNetowrk* (interception and caching) within their SDK. To implement the interception, the *Ad Network* can redirect all the advertising requests to the background service on the mobile phone. The server side of *eeAdNetwork* can be added to the existing *Ad Network* servers that can easily share information (for e.g., user's-context, advertisement preference, etc.) with the *eeAdNetwork* mobile component.

Mobile operators can integrate *eeAdNetwork* with their current advertisement platform. The necessary modifications (`/etc/hosts` and IPTABLE) for interception and redirection of advertisement traffic on the mobile device can be configured by the operator before selling the device to the user. The server side of *eeAdNetwork* can work as a gateway as detailed in this thesis or integrated into their *Ad Network*.

Lastly, *eeAdNetwork* can be run by a third party or by the user's themselves. Users will need to download and install the *eeAdNetwork* application (running the background service, cache and communication to the *Ad Network*) from the application store. To make the intercepting work, users will have to root their devices [21]. The server side service can run in the public cloud (e.g., Amazon EC2) and communicate directly with the existing *Ad Networks* (e.g., AdMob).

Chapter 7

Conclusions

In this thesis, we discuss the impact of mobile phone advertisements on the phone's battery. We focus on mobile applications that use the Internet connection mainly for fetching advertisements, for example, mobile arcade games do not necessarily require Internet connectivity and only use it to fetch advertisements. Currently, applications use a fixed interval to fetch advertisements, typically, at 30s, 60s, 90s, 120s intervals. We investigate the power consumed by advertisements in 5 popular mobile games and observe that the advertisements consume from 9%–27% of the application's total power consumption. The variation in power consumption of the different applications is due to the use of different refresh intervals.

Since, accessing advertisements over 3G and WiFi interfaces should result in different amount of power consumption, we observe that delivering advertisements at fixed intervals can consume up to 27% of the application's total energy consumption over 3G and 15% over WiFi. We also observe that the advertisement server routinely responds with the same advertisement, which suggests that the application is needlessly requesting for more advertisements. Consequently, we observe that the advertisement server sends roughly 8 unique advertisements/hour, i.e., an application requesting advertisements at *30s* intervals receives, sends 120 requests in an hour but receives only 8 unique advertisements.

To reduce the energy consumption, we propose and implement a *eeAd-*

Network, a transparent cache that intercepts the advertisement requests and caches the response on the mobile phone. Thereafter, all forthcoming requests are served from the local cache. The number of advertisements cached in the *eeAdNetwork* is configurable, but our implementation caches 8 new advertisements/hour. At the end hour, the cache is refreshed with new advertisements from the advertisement server. Our results show that by using the *eeAdNetwork* the share of the power consumption for delivering advertisements drops from 25% to $\approx 4\%$.

These are preliminary results and are focused on a custom made application, for more detailed analyses the experiments with *eeAdNetwork* should be run on mobile games published on the market. This is left as a task for future work.

Bibliography

[1] Armand, M., and Tarascon, J.-M. Building better batteries. *Nature 451*, 7179 (2008), 652–657.

[2] Balasubramanian, N., Balasubramanian, A., and Venkataramani, A. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (2009), ACM, pp. 280–293.

[3] Bureau, I. A. Iab platfrom status report: A mobile advertising overview. webpage, July 2008. `http://www.iab.net/media/file/moble_platform_status_report.pdf/`. Accessed 13.3.2013.

[4] Carroll, A., and Heiser, G. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference* (2010), pp. 21–21.

[5] Chin, E., Felt, A. P., Greenwood, K., and Wagner, D. Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services* (2011), ACM, pp. 239–252.

[6] Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation* (2010), pp. 1–6.

[7] Falaki, H., Lymberopoulos, D., Mahajan, R., Kandula, S., and Estrin, D. A first look at traffic on smartphones. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (2010), ACM, pp. 281–287.

[8] Favell, A. mobithinking guide to mobile advertising networks (2012). webpage, 2012. `http://mobithinking.com/mobile-ad-network-guide/`. Accessed 13.3.2013.

[9] Felt, A. P., Chin, E., Hanna, S., Song, D., and Wagner, D. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security* (2011), ACM, pp. 627–638.

[10] Gartner, Inc. Gartner says free apps will account for nearly 90 percent of total mobile app store downloads in 2012, 2012. `http://www.gartner.com/newsroom/id/2153215`. Accessed 20.6.2013.

[11] Grace, M. C., Zhou, W., Jiang, X., and Sadeghi, A.-R. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks* (2012), ACM, pp. 101–112.

[12] Guha, S., Cheng, B., and Francis, P. Challenges in measuring online advertising systems. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (2010), ACM, pp. 81–87.

[13] Guha, S., Reznichenko, A., Tang, K., Haddadi, H., and Francis, P. Serving ads from localhost for performance, privacy, and profit. In *Proceedings of the 8th Workshop on Hot Topics in Networks (HotNets? 09), New York, NY* (2009).

[14] Haddadi, H., Hui, P., and Brown, I. Mobiad: private and scalable mobile advertising. In *Proceedings of the fifth ACM international workshop on Mobility in the evolving internet architecture* (New York, NY, USA, 2010), MobiArch '10, ACM, pp. 33–38.

[15] Inc., G. Worldwide Mobile Phone Sales. `http://www.gartner.com/newsroom/id/2335616`. Accessed 10.04.2013.

[16] Khan, A. J., Subbaraju, V., Misra, A., and Seshan, S. Mitigating the true cost of advertisement-supported free mobile applications. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications* (2012), ACM, p. 1.

[17] Leontiadis, I., Efstratiou, C., Picone, M., and Mascolo, C. Don't kill my ads!: balancing privacy in an ad-supported mobile application market. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications* (New York, NY, USA, 2012), HotMobile '12, ACM, pp. 2:1–2:6.

[18] Miniclip.com. Fragger. `https://play.google.com/store/apps/details?id=com.miniclip.fraggerfree`.

[19] Mobile Marketing Association (MMA. Mobile Advertising Guidelines. `http://www.mmaglobal.com/files/mmaglobal.com/file/mobileadvertising.pdf`. Accessed 10.04.2013.

[20] Monsoon Solutions Inc. Mobile Device Power Monitor Manual. `http://msoon.github.io/powermonitor/PowerTool/doc/Power%20Monitor%20Manual.pdf`. Accessed 10.04.2013.

[21] Nexus S Hacks. How to Root Nexus S and Nexus S 4G on ICS, Gingerbread, or Jelly Bean! `http://nexusshacks.com/nexus-s-root/how-to-root-nexus-s-or-nexus-s-4g-on-ics-or-gingerbread/#comment-6216`. Accessed 10.04.2013.

[22] nixCraft. Linux iptables: Port Redirection. `http://www.cyberciti.biz/faq/linux-port-redirection-with-iptables/`.

[23] Pathak, A., Hu, Y. C., and Zhang, M. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems* (2012), ACM, pp. 29–42.

[24] Pathak, A., Hu, Y. C., Zhang, M., Bahl, P., and Wang, Y.-M. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems* (2011), ACM, pp. 153–168.

[25] Pearce, P., Felt, A. P., Nunez, G., and Wagner, D. Addroid: Privilege separation for applications and advertisers in android. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security* (2012), ACM, pp. 71–72.

[26] Perrucci, G., Fitzek, F., and Widmer, J. Survey on energy consumption entities on the smartphone platform. In *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd* (2011), IEEE, pp. 1–6.

[27] Prochkova, I., Singh, V., and Nurminen, J. K. Energy cost of advertisements in mobile games on the android platform. In *Next Generation Mobile Applications, Services and Technologies (NGMAST), 2012 6th International Conference on* (2012), IEEE, pp. 147–152.

[28] Qian, F., Wang, Z., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. Characterizing radio resource allocation for 3g networks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2010), IMC '10, ACM, pp. 137–150.

[29] Rovio Mobile. Angry Birds. `https://play.google.com/store/apps/details?id=com.rovio.angrybirds`.

[30] Runnergames. Ceramic Destroyer. `https://play.google.com/store/apps/details?id=com.game.CeramicDestroyer`.

[31] Runnergames. Skater Boy. `https://play.google.com/store/apps/details?id=com.game.SkaterBoy`.

[32] Runnergames. Yoo Ninja! Free. `https://play.google.com/store/apps/details?id=com.RunnerGames.game.YooNinja_Lite`.

[33] Samsung. Nexus S Android Smartphone. `http://www.samsung.com/us/mobile/cell-phones/GT-I9020FSTTMB`. Accessed 10.04.2013.

[34] Smaato. The mobile advertising ecosystem. webpage, August 2010. `http://www.smaato.com/media/Smaato_WhitePaper_Mobile_Advertising_Ecosystem_082010.pdf/`. Accessed 13.3.2013.

[35] Thiagarajan, N., Aggarwal, G., Nicoara, A., Boneh, D., and Singh, J. P. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web* (2012), ACM, pp. 41–50.

[36] Vallina-Rodriguez, N., and Crowcroft, J. Energy management techniques in modern mobile handsets. *Communications Surveys & Tutorials, IEEE*, 99 (2012), 1–20.

[37] Vallina-Rodriguez, N., Shah, J., Finamore, A., Grunenberger, Y., Papagiannaki, K., Haddadi, H., and Crowcroft, J. Breaking for commercials: Characterizing mobile advertising. In *Proceedings of the 2012 ACM conference on Internet measurement conference* (2012), ACM, pp. 343–356.

[38] Wikipedia. Targeted Advertising. `http://en.wikipedia.org/wiki/Targeted_advertising`. Accessed 10.04.2013.

[39] Zhang, L., Gupta, D., and Mohapatra, P. How expensive are free smartphone apps? *SIGMOBILE Mob. Comput. Commun. Rev. 16*, 3 (Dec. 2012), 21–32.

Appendix A

Source Code

The source code of the Android mobile application and the server side application can be found at `https://github.com/iprockova/ProxyServer` and `https://github.com/iprockova/RelayServer`, respectivly. Android Support library[1] is required to run the mobile application.

[1] http://developer.android.com/tools/extras/support-library.html