Prefetching mobile ads: Can advertising systems afford it?

Prashanth Mohan [*]
UC Berkeley

Suman Nath Oriana Riva
Microsoft Research

Abstract

Mobile app marketplaces are dominated by free apps that rely on advertising for their revenue. These apps place increased demands on the already limited battery lifetime of modern phones. For example, in the top 15 free Windows Phone apps, we found in-app advertising contributes to 65% of the app's total communication energy (or 23% of the app's total energy). Despite their small size, downloading ads each time an app is started and at regular refresh intervals forces the network radio to be continuously woken up, thus leading to a high energy overhead, so-called 'tail energy' problem. A straightforward mechanism to lower this overhead is to prefetch ads in bulk and serve them locally. However, the prefetching of ads is at odds with the real-time nature of modern advertising systems wherein ads are sold through real-time auctions each time the client can display an ad.

This paper addresses the challenge of supporting ad prefetching with minimal changes to the existing advertising architecture. We build client models predicting how many ad slots are likely to be available in the future. Based on this (unreliable) estimate, ad servers make client ad slots available in the ad exchange auctions even before they can be displayed. In order to display the ads within a short deadline, ads are probabilistically replicated across clients, using an overbooking model designed to ensure that ads are shown before their deadline expires (SLA violation rate) and are shown no more than required (revenue loss). With traces of over 1,700 iPhone and Windows Phone users, we show that our approach can reduce the ad energy overhead by over 50% with a negligible revenue loss and SLA violation rate.

[*] Work done as an intern at Microsoft Research.

Eurosys'13 April 15-17, 2013, Prague, Czech Republic
Copyright © 2013 ACM 978-1-4503-1994-2/13/04. . . $15.00

Categories and Subject Descriptors J.7 [*Computer Applications*]: Computers in other systems.

General Terms Algorithms, Design, Economics, Performance.

Keywords Internet advertising, Smartphones, Energy, Real time markets.

1. Introduction

The consumer draw of free mobile applications (apps) over paid apps is prevalent in app stores such as the Apple App Store, the Google Play market and the Windows Phone (app) Store. The Google Play market, for instance, reports roughly 2/3 of their apps as being free [3] of which the large majority (80% according to a recent study [20]) rely on targeted advertising as their main business model.

While the use of in-app advertising has enabled the proliferation of free apps, fetching and displaying advertisements (ads) in an application significantly contributes to the application's energy consumption. In Section 2, we quantify this cost for popular Windows Phone apps and find that on average ads consume 65% of an app's total communication energy, or 23% of an app's total energy.

Mobile apps refresh their ads every 12–120 seconds [12, 37]. Just downloading an ad takes no more than a few seconds. However, after an ad's download is completed, the 3G connection stays open for an extra time, called 'tail time', which, depending on the network operator, may be 10 (Sprint 3G) or even 17 (AT&T 3G) seconds [25]. The tail time alleviates the delay incurred when moving from the idle to the high-power transmission state of the device's radio, but at the cost of energy, so-called 'tail energy'. This results in the high energy overhead incurred by ads.

Previous work has investigated energy-saving mechanisms to address the tail energy problem (see Section 6 for more details). In general, most of these solutions study how to adapt the tail time [9, 39], but as this parameter is under the network operator's control they require modifications of the base station, thus incurring significant deployment obstacles. Other energy-saving solutions, such as switching from low-power/low-bandwidth interfaces to high-power/high-bandwidth interfaces when network activity requires [29], involve modifications at the device's network stack. A last approach, which we adopt, is prefetching (or batching of data transmissions in general) [7]. This can be done at the application level, without requiring any changes in network operators or device OS. Ads are prefetched in bulk, periodically or when network conditions are favorable. The mobile platform serves ads locally, until the budget is exhausted or the cache is invalidated (e.g., ads have expired).

Achieving an accurate prefetching model is hard in general, but if a good prediction model can be built, integrating the concept of prefetching into an actual application is usually straightforward. To achieve high energy savings, the key is to prefetch as much content as possible in batches, perhaps once or twice a day (e.g., in the morning when the user wakes up). Prefetching can be managed by the client, independently of the server. While this is all true for many applications, particularly web browsing [4, 22, 27] and web search [7, 19], it is not practical for advertising systems. In these systems, ads are sold through on-demand auctions, and only if a client can immediately display an ad. By offering the client the option of downloading ads in batches, an advertising system risks losing revenue, especially for display ads where the advertising system pays based on ad impression count. Revenue loss can occur either because a prefetched ad is not shown within its deadline or is shown more times than required (this can happen if the ad is prefetched at the same time by multiple clients to ensure it will be shown eventually). Despite the risk, as the largest mobile advertising platforms are owned by the largest smartphone providers (e.g., Google's AdMob and Android, Apple's iAds and iPhone, Microsoft's MSN Ad Network and Windows Phone), there is an incentive to provide low-energy apps to attract users.

Modern advertising systems include an ad server that mediates between client devices and ad exchanges. Ad exchanges are neutral parties that aggregate ads from different third-party ad networks. One way to integrate ad prefetching into current advertising systems is to expose to the ad networks the uncertainty of an ad being shown on the client devices, and charge advertisers only if the ad is actually shown. The advantage is that ad networks can directly manage their risk by balancing with their ad campaigns. The disadvantage is that ad networks' decisions are limited to their own inventory and lack global knowledge. More importantly, this approach significantly alters the existing ad exchange model and makes the interface to advertisers more complex. Instead, we decide to favor ease of deployability and limit changes to the current infrastructure.

This paper explores to what extent today's advertising systems can afford, with minimal changes to their infrastructure, to offer ad prefetching to mobile clients, how much energy can be saved, and what risk prefetching entails. We start by considering the problem of predicting for how long a certain user is likely to use phone applications in a given time interval. This prediction gives an indication of how many ads the client will be able to consume in the future. Note that ad platforms today already have access to logs of their clients, so this prediction comes at no additional privacy cost. We analyze the application traces of 1,693 Windows Phone users over one month and 25 iPhone users over one year, and measure the predictability of app usage from past behavior. Our entropy-based evaluation shows that user-specific and time-dependent prediction models are the most accurate among those we considered, but can still be somewhat inaccurate. The client may fetch too few ads, thus achieving only limited energy savings. Or the client may fetch too many ads and not be able to display all of them by their deadlines, thus causing SLA violations for the ad infrastructure. We compensate for the inaccuracy of the prediction model by scheduling ads in order of expiring deadlines and by introducing an overbooking model that probabilistically replicates ads across clients. Our evaluation shows that our approach is capable of reducing the energy consumed by an average client by 50%, while maintaining SLA violation rates below 3%.

1.1 Contributions and overview

Prefetching has been used to save energy in various contexts [4, 7, 19, 22, 27], and particularly the idea of prefetching ads on mobile phones has been previously considered [13, 17, 35, 37]. However, we are not aware of any work that looked at the problem of ad prefetching in detail, and produced a solution compatible with existing ad infrastructures. Recent work on privacy-preserving advertising implicitly assumes prefetching of ads in bulk. For instance, in Privad [13], each client subscribes to coarse-grained classes of ads the user is interested in and an anonymization proxy constantly fetches ads on the client's behalf. Our design is compatible with such systems and can help them become a feasible business model for privacy preservation.

In summary, this paper makes the following contributions: (i) proposes a methodology for accurately measuring the energy overhead of mobile ads and gives an estimate of such an overhead based on popular Windows Phone apps; (ii) studies the predictability of app usage behavior and derives personalized, time-based models based on roughly 1,700 iPhone and Windows Phone user traces; (iii) models the problem of ad prefetching as an overbooking problem where the ad server can explicitly tune the risk of SLA violations and revenue loss; and (iv) evaluates the feasibility of the proposed approach and quantifies the energy savings for a realistic population of mobile users.

The rest of the paper is organized as follows. In the next section, we motivate this work and report how much energy ads consume in 15 of the most popular Windows Phone apps. Section 3 illustrates how existing mobile advertising systems work and explains why it is hard to add support for ad prefetching in bulk. The next two sections present and evaluate our approach including app usage prediction and overbooking model. We review related work in Section 6 and conclude in Section 7.

2. The real cost of mobile ads

More than half of the apps available in today's app marketplaces are ad-supported [20]. How much energy do these ad-supported apps consume to communicate with ad servers? In this section, we empirically answer this question with the top 15 ad-supported Windows Phone apps. Note that some of the most popular apps (e.g., Facebook, YouTube) do not display any ads, while some other popular apps (such as Angry Birds) are ad-free paid apps; they are omitted from our study.

2.1 Communication costs for serving ads

Previous work highlighted significant overheads of ads in smartphone apps [17, 28, 37]. In [17], the authors measured non-negligible network traffic due to ads in Android apps. Translating network traffic into communication energy, however, is nontrivial because communication energy depends on various other factors such as the current radio state, radio power model, radio bandwidth, etc. In [28], the authors propose *eprof*, a fine-grained energy profiler for smartphone apps that traces various system calls and uses power models to report energy consumption of various components of an app. Using this tool, the authors demonstrate that third party modules consume significant energy on six Android apps they study (e.g., Flurry [1], a third party data aggregator and ad generator, consumes 45% energy in the AngryBirds application). The goal of the study was not to isolate the communication overhead of ad modules. In fact, with *eprof*'s approach, accurately isolating communication overhead of a third party module alone is nontrivial when the app itself communicates with a backend server and communications of the app and third-party module interleave. This is because when multiple components share a radio for communication, it is not clear to whom to attribute the nontrivial wake-up and tail energy of the radio.

2.2 Measurement methodology

In order to isolate the exact communication overhead of ads within an app, we use an approach different from *eprof* [28]. Given an app, we produce three versions of it: the original *ad-enabled* version, a modified *ad-disabled* version that does not communicate with the ad server and shows a locally cached ad, and a modified *ad-removed* version that does not show any ad. We then execute all these versions with the same user interaction workload and measure their energy consumption. The difference between the first and the second version gives the communication energy overhead due to the ad module, and that between the second and the third version gives the non-communication energy overhead of the ad module. This approach is more accurate than *eprof* as we do not need to use any ad-hoc heuristics to distribute shared network costs between the app and ad modules. However, in taking this approach we need to address several challenges.

Measuring energy. To compare the ad module's communication energy with that of the app, we need to measure their communication energy as well as total energy. Thus, tools that give only total energy (such as a powermeter or battery level monitor) are not sufficient. We use WattsOn [25], a tool for estimating energy consumption of a Windows Phone app while running it on the Microsoft's Windows Phone Emulator (WPE). WPE can run any app binary downloaded from the Windows Phone Marketplace. When an app runs on WPE, WattsOn captures all network activities of the app and uses state-of-the-art power models of WiFi, 3G radio, CPU, and display to estimate communication and total energy of the app. WattsOn also allows using various operating conditions (carrier, signal strength, screen brightness). Experiments on a Samsung Focus Windows Phone show that WattsOn's estimation error is $< 5\%$ for networked apps, compared to the true energy measured by a power meter.

In our measurements, WattsOn is set to simulate the Samsung Focus phone with AT&T as the carrier. The phone uses 3G communication and enjoys 'good' signal strength and network quality, with average download and upload bandwidth of 2500 kbps and 1600 kbps respectively [34]. The display is configured with medium brightness.

Producing ad-disabled and ad-removed versions of an app. To produce an ad-disabled version of a given app, we need to disable communication of ad modules with ad servers. This is relatively simple for most of the apps we tried: these apps include standard ad controls (such as Windows Ad Control or AdDuplex) that communicate with predefined ad servers. To disable such communication, we redirect application DNS requests for their ad servers to the localhost interface (127.0.0.1). WattsOn ignores any communication redirected to this interface. As we run the apps in an emulator, this is done easily by modifying the `/etc/hosts` file in the machine the WPE runs on. Most ad controls show a default locally-cached ad after failing to connect to ad servers, without affecting an app's normal operations.

The above simple trick does not work for apps (e.g., BubbleBursts) that dynamically download IP addresses of ad servers from the network. It also does not work for apps (e.g., Weather) that use the same backend server for downloading ads and useful application data. For such apps, we used binary rewriting techniques to modify app binaries to remove instructions that request new ads. Windows Phone apps are written in .Net, and we used the Common Compiler Infrastructure library [24] for such rewriting.

To generate an ad-removed version of an app, we use binary instrumentation to replace the app's ad module with a dummy module that does not do anything other than showing a blank box.

Workload. We use each app three times and report the average energy consumption. In each run of an app, we use it for two minutes in its expected usage mode. For example, if the app is a game, we start it and play a few levels of the game;

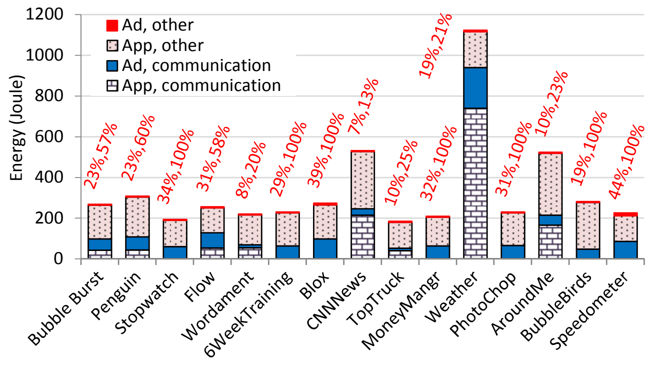


Figure 1. Energy consumed by top ad-supported WP apps. Both original and ad-disabled versions are run with the same sequence of user interactions for 2 minutes. The label $x\%, y\%$ on top of each bar means ads consume $x\%$ of the total energy and $y\%$ of the communication energy of the app. Ads consume significant communication energy, even for communication-heavy apps. *On average, ads consume* 23% *of the total energy, or* 65% *of the communication energy, of an app.*

if the app is a news app, we open it, navigate to a few pages and read them.

A typical app's energy consumption depends on how it is used, e.g., which pages within the app are visited and for how long. For a fair comparison of the ad-enabled, ad-disabled, and ad-removed versions of the same app, we run them with the same sequences of user interactions. More precisely, we record user interactions with the original app and replay the recorded interactions on all three versions of the app. Some apps show random behavior across runs. For example, the Bubble Burst game starts with a random game every time it is started. For such apps, we cannot replay a prerecorded sequence of user interactions, and hence we simply use all three versions independently for the same duration of time.

2.3 Results

Figure 1 shows the results of our measurements for the top 15 ad-supported Windows Phone apps. These apps use various ad controls such as Windows Phone Ad Control, AdMob, AdDuplex, Somaata, and DoubleClick, and some apps use multiple ad controls.

Our measurements reveal several important points. First, ad modules consume a significant part of an app's energy. Across the 15 apps we measured, ads are, on average, responsible for 23% of the total energy consumed by the app (including CPU, display, and communication), and 65% of the total communication energy. This overhead is significant, considering that typical ads are small in size ($<$ 100 bytes for most textual ads). Second, the overhead of ads is bigger in apps such as Blox and Speedometer with no or small network activity.

On the other hand, in apps such as CNNNews and Weather that need to communicate with the Internet, communication of an ad module can often piggyback on the phone's already-turned-on radio. Interestingly, overheads of ads are substantial even in communication-heavy apps such as Weather and CNNNews—without ads, these apps can keep the phone's radio in low power state more often. Third, most of the overhead of ad modules comes from communication: CPU and display constitute $< 8\%$ of the total ad overhead. We therefore focus on reducing communication overheads of ads.

2.4 Tail energy problem

Why do in-app ads consume more than half of the communication energy consumed by the app itself? Typically, in a GSM or 3G network, the radio operates in three power states: 'idle' if there is no network activity; DCH (Dedicated Channel) in which a channel is reserved and high-throughput, low-delay transmission is guaranteed; and FACH (Forward Access Channel) in which a channel is shared with other devices and is used when there is little network traffic. The idle state consumes 1% of the power of the DCH state, and the FACH state consumes about half of the DCH power. After a transmission, instead of transitioning from the DCH state to the idle state, the device spends some extra time in the DCH state and then in the FACH state—5 and 12 seconds respectively for an AT&T 3G network [25]. This delay, called 'tail time', determines how responsive the device is when new network activity starts. The tail time cannot be controlled by the application directly. Instead, each network provider decides on this tradeoff: a longer tail time consumes more power, but makes the device more responsive; a shorter tail time consumes less power, but introduces delays [9].

Each time a user starts ad-supported mobile apps, ads are fetched one by one, and they are regularly refreshed during app operation. Downloading an ad takes no more than a few seconds, but once the ad's download has completed, the 3G connection stays open for the extra tail time. The energy consumed during this idle time, called 'tail energy', causes the ads' large energy overhead. In a typical 3G network (with a tail time of 12.5 seconds), Balasubramanian et al. [7] have shown that about 60% of the energy consumed for a 50 kB download is tail energy. The overhead is even bigger for shorter downloads, such as a typical ad of size 5 kB.

The overall severity of the tail energy problem may depend on various factors. One factor is the usage of WiFi networks instead of cellular networks. WiFi's tail energy problem is smaller (tail time is $\approx$ 500ms). To keep our study simple, we focus on only one type of network and we prefer cellular networks over WiFi for two main reasons. First, several recent studies have pointed out that smartphone users spend significant fraction of their app-usage time (reported to be $> 90\%$ in [15] and $> 60\%$ in [8]) on cellular networks compared to WiFi networks.[1] Moreover, 3G coverage is significantly larger than WiFi coverage (e.g., reported to be $8\times$ more in US cities [6]) In fact, a non-negligible fraction of users (e.g., reported to be $> 25\%$ in a survey [11]) do not even enable WiFi connectivity even though their phones are WiFi-capable. Finally, WiFi is more energy dense than 3G (i.e., the WiFi radio consumes more energy for each second it is switched *on* compared to the 3G radio), making WiFi unsuitable for short transfers such as ads [30].

Another factor that might affect the overall impact of the tail energy is the presence of background traffic. Unfortunately, applications' traffic is unlikely to significantly alleviate the tail energy problem of ads traffic. We observed this in our experiments with the Weather and CNNNews apps, and a recent measurement study [37] based on the one-day data connections of more than 3 million subscribers of a major European mobile network confirms this finding. The study reports that roughly 81% and 68% of the network traffic related to ads was isolated for Android and iPhone devices respectively. In most cases, interleave times between network activity sessions (of apps themselves and ads) were so high that the device radio was likely to be in the idle power mode when ad requests were generated (We experimentally verify the impact of background traffic in our evaluation).

In the rest of the paper, we investigate how prefetching can help reduce the high energy overhead of ads. Intuitively, prefetching can amortize the tail energy cost among multiple ads. Our experiments show that downloading 10 ads of size 1 kB (or 5 kB) in bulk over AT&T 3G network consumes $8.6\times$ (or $4.1\times$ respectively) less energy than downloading them one every minute. Moreover, prefetching enables downloading ads at opportune times, such as when the phone is being charged or WiFi connectivity is available. Finally, unlike solutions that require changes at the networking stack, a prefetching-based solution can be deployed on today's smartphones with minimal changes to existing ad infrastructures, as we will show later.

3. Prefetching ads: feasibility and challenges

We start by providing a description of how mobile advertising works today and then discuss the challenges involved in supporting batch prefetching of ads with minimal changes to the current infrastructure.

3.1 Background on mobile advertising

As Figure 2 shows, a typical mobile advertising system consists of five parties: mobile clients, advertisers, ad servers, ad exchanges and ad networks. A mobile application includes an ad control module (e.g., AdControl for Windows Phones, AdMob for Android) which notifies the associated ad server any time an *ad slot* becomes available on the client's device.

[1] The volume of data traffic over WiFi, however, is reported to be bigger than that over cellular network.

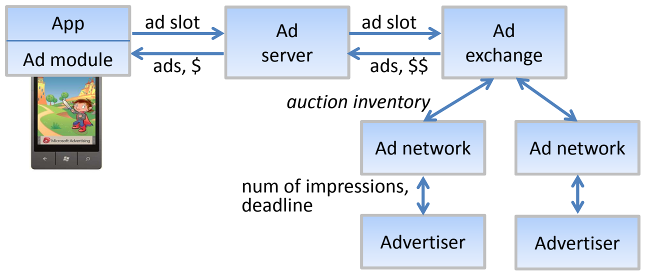


Figure 2. Architecture of a typical mobile ad system.

The ad server decides how to monetize the ad slot by displaying an ad. Ads are collected from an *ad exchange*. Ad exchanges are neutral parties that aggregate ads from different third party ad networks and hold an auction every time a client's ad slot becomes available. The ad networks participating in the exchange estimate their expected revenue from showing an ad in such an ad slot and place a bid on behalf of their customers (i.e., the advertisers). An ad network attempts to maximize its revenue by choosing ads that are most appropriate given the context of the user, in order to maximize the possibility of the user clicking on the ads. The ad network receives information about the user such as his profile, context, and device type from the ad server, through the ad exchange. Ad exchange runs the auction and chooses the winner with the highest bid.

Advertisers register with their ad networks by submitting an *ad campaign*. A campaign typically specifies an advertising budget and a target number of impressions/clicks within a certain deadline (e.g., 50,000 impressions delivered in 2 weeks). They can also specify a maximum cap on how many times a single client can see a specific ad and how to distribute ads over time (e.g., 150 impressions per hour).

The ad server is responsible for tracking which ads are displayed and clicked, and thus determining how much money an advertiser owes. The revenue of an ad slot can be measured in several ways, most often by views (Cost Per Impression) or click-through (Cost Per Click), the former being most common in mobile systems. The ad server receives a premium on the sale of each ad slot, part of which is passed to the developer of the app where the ad was displayed.

3.2 A proxy-based ad prefetching system

One way to incorporate ad prefetching into the existing ad ecosystem is to use a proxy between the ad server and the mobile client. A client with available ad slots contacts the proxy that prefetches a batch of ads from the ad exchange (through the ad server) and sends the batch to the client. After the client has displayed all ads of the batch, it contacts the proxy again and gets the next batch of ads. Such a solution is easy to implement in any existing smartphone app that receives ads through an ad control module embedded within the app. The client-side of the prefetching logic can be implemented within the ad control, while the server-side

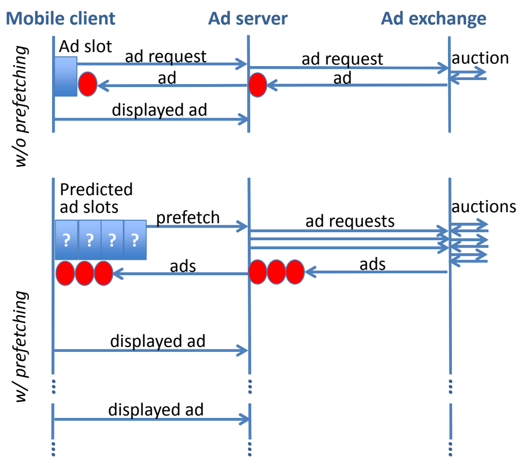


Figure 3. Ad system without and with ad prefetching (the prefetching proxy runs at the ad server).

logic can be implemented in the ad server, which acts as the prefetching proxy. Figure 3 shows how the ad system works today without ad prefetching, and how it can work with ad prefetching.

While it is easy to incorporate a prefetching proxy into the existing ad ecosystem, feasibility and advantage of prefetching depend on various factors. The two key decisions a prefetching model must take are 'what' to prefetch and 'when' to prefetch. These have been successfully addressed in the context of web browsing [4, 22, 27] and web search [19]. However, the key property that distinguishes ad prefetching from other prefetching scenarios is that *ads have deadlines and there are penalties if ads are prefetched, but not served within their deadlines*. In fact, ad prefetching cannot be implemented as a stand-alone client action as for web browsing. A bad prefetching strategy that does not respect ad deadlines can adversely affect other parties involved in the ad ecosystem.

Ad deadlines. The deadline D of an ad may come from multiple sources. The advertiser typically wants all ads in an ad campaign to be served within a deadline. For example, an advertiser may start a campaign for an upcoming sale and the associated ads must be delivered before the sale ends. Even when an advertiser puts a long deadline, it expects the ad network to guarantee some SLA about the rate at which ads are served. For example, an advertiser may start a one-week ad campaign, but still want the guarantee that 100 impressions of its ads will be served per hour. In existing ad systems, these are the only factors affecting ads' deadlines since ads are delivered to clients within a short time period time (few hundred milliseconds) after being retrieved from the ad exchange.

With prefetching, however, other factors play into deciding an ad's deadline. Suppose the proxy serves ads within a serving period (smaller than the ad deadline specified by the advertiser). Since the bid price for the ads changes over time,

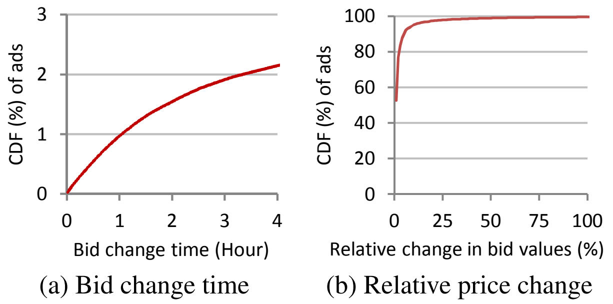


Figure 4. CDFs of (a) how often bid prices change for an ad and (b) relative price difference when a bid price changes.

the proxy (hence, the ad server) takes some risks in prefetching ads. One type of risk is that it may not be able to serve all prefetched ads within the advertiser's deadline. We will discuss this risk in more details later. Another type of risk is that the bid price for an ad may change within the serving period, and if the price actually goes up, the ad server could have made more revenue by collecting the ad from the ad exchange at a later point in time rather than at the beginning of the serving period. How big is this risk?

To answer this, we sampled around 1 TB of log data from a production advertising platform spanning a 12 hour period in August 1, 2012. The data covers over several hundred million auctions and unique ads shown to users of a popular search engine across all search topics. The trace record for an auction lists all the ads that participated in it (whether the ad was ultimately shown or not) and the bids corresponding to each ad. We use this trace to understand how often the bid price for an ad changes and by how much. We observed that most ads do not change their bid prices within the trace period. Figure 4(a) shows a portion of the CDF of the average time of an ad changing its bid price. As shown, only less than 0.5% of the ads change their bid price within 30 minutes of their most recent price change (or their first appearance in the system). Figure 4(b) shows the CDF of the relative price change when an ad changes its bid: 95% of the bid changes are within 10% of the previous bid price.

The results above highlight that even though auction prices in the ad exchange are dynamic, the dynamics and hence the revenue risks of the ad server are relatively small for a small window of time. For example, if the ad server prefetches ads for a serving period of 30 minutes, the probability that any of the ads will change its bid price within the serving period is $< 0.5\%$, and even if an ad changes its bid price, the change will be $< 10\%$ in 95% of the cases. An ad server can choose a suitable value for its serving deadline depending on the dynamics. Unless otherwise specified, we assume a serving deadline of 30 minutes in the rest of the paper.

With prefetching, the actual deadline of an ad is the minimum of the deadline specified by the advertiser and the serving deadline the server uses for its prefetched ads. Deadlines specified by advertisers are typically longer than 30 minutes, and therefore, we consider the serving deadline as the deadline D for all prefetched ads.

3.3 Ad prefetching tradeoffs

Ideally, a proxy would want to serve all prefetched ads within their deadline D. This is possible if the proxy can predict exactly how many ads it will be able to serve to its clients within its prediction period. As we will show later, however, such predictions are unlikely to be accurate in practice. Hence, the ad infrastructure runs into the following two types of risks.

- **SLA violations:** SLA violations may happen if a proxy (hence, the ad server) fails to deliver a prefetched ad within its deadline (e.g., an ad for a promotion is displayed after the promotion ends, or an ad is displayed when its bid price is significantly different from when it was prefetched).
- **Revenue loss:** Revenue earned by an ad server is related to the number of unique ads served to clients. A bad ad-prefetching and -serving system can cause revenue loss to the ad server: by serving ads after deadlines, by serving the same ad impression to more users than required, by not serving an ad even when a slot is available, etc.

There exist tradeoffs between the above two factors and network overhead. An aggressive strategy may prefetch more ads than can be served to the client within the ad deadline— this will minimize communication overhead, but cause frequent SLA violations. On the other hand, a conservative prefetching approach may avoid SLA violations by prefetching a small number of ads, but will incur large communication overhead. The prefetching proxy may consider replicating ads to reduce SLA violations: it can send the same ad impression to multiple clients to maximize the probability that the ad impression will be served to at least one client. However, this may incur revenue loss. A good prefetching strategy needs to strike a good balance between these competing factors.

In the rest of the paper, we will investigate these tradeoffs. In particular, we consider the following mechanisms, implemented in a proxy, and their effects on energy, SLA violations and revenue.

- **App usage prediction:** The prefetching proxy estimates how many ads a client will be able to show in next T minutes (T is called the *prediction interval*) and prefetches that many ads from the ad server. A perfect prediction should result in an optimal communication energy, with no SLA violations and no revenue loss.
- **Overbooking:** The proxy may replicate one ad across multiple clients. In the presence of inaccurate predictions, this may reduce SLA violations, but increase revenue loss.

4. Ad prefetching with app usage prediction

The first challenge to address is to decide how many ads to prefetch and how often. Suppose each ad comes with a deadline of D minutes, all ads are of the same type[2] and one ad is displayed every t minutes during app usage. We also call t the size of an ad slot and the refresh period of an ad. For simplicity, let us assume for now that the client periodically prefetches ads once every round of T minutes (the prediction period). If the client could predict the number of ad slots (k) available in the next round, it could prefetch exactly k ads, satisfying the client's needs and without wasting any ads. *Is this kind of accurate prediction possible in practice?*

4.1 App usage prediction

The number of ad slots available in the future depends on how often the user is likely to use apps installed on his phone. We analyze two real user datasets to answer the following key questions:

1. Is app usage predictable based on users' past behavior?
2. What features of past app usage are useful in prediction?

Note that previous work considered predicting what apps will be used in a given context in order to preload apps on the phone [38] or to customize homescreen icons [32]. In contrast, we aim to find *how long* apps will be used in a given time window.

Datasets. We use the following two datasets reporting mobile applications' usage.

- Windows Phone logs: we use the device logs of 1,693 WP users over roughly a month. Users were randomly selected worldwide, among a larger number of mobile users that opted into feedback.
- iPhone logs: we use the device logs of 25 iPhone users [21, 31]. Logs were collected by the LiveLab project at Rice University. The deployment involved 25 undergraduate iPhone users and lasted one year.

We filtered the logs by eliminating apps which do not support ads, such as call application, sms, alarm clock, and settings. We then assumed all remaining apps display an ad at startup time and refresh it every t minutes, where $t = 1$.

► **Predictability with past behavior.** We first use an information-theoretic measure to get insights into predictability of phone usage based on past behavior. The measure tells us *how often* we can predict phone usage. Even though this does not tell us *how accurately* we can predict or *how* we can do reasonable prediction, the analysis gives us valuable insights that we use in the prediction process we describe later.

Information entropy is a well-known metric that measures the level of uncertainty associated with a random process. It quantifies the information contained in a message, usually in

[2] The ad type indicates the app category the ad was initially bid for.

bits/symbol. The entropy of a discrete random variable X is defined as

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

where $p(x)$ is the probability mass function, $0 \leq p(x) \leq 1$. In our scenario, the variable X denotes the value of k (the number of ad slots) in a given round of length T (the prediction period).

To understand how predictable X is in our datasets, we compute the entropy of the underlying process that determines the value of X. From a given dataset, we compute the PDF of X, with $\Pr(X = i)$ as the probability of having i ad slots in time T (i.e., the probability of the user using apps for i ad slots in a window of T minutes). Finally, we compute the entropy of X by using the above equation. For concreteness, assume that T = 60 minutes. The value of k can be any integer within the range $[0, 60]$. Thus, the value $\log_2(61) \approx 6$ gives the upper bound of X's entropy.

Since entropy tells us about the uncertainty associated with a process, it can implicitly provide information about its predictability. When the entropy is 0, the outcome of the process is completely deterministic and hence completely predictable. On the other hand, when the process is completely random, $p(x)$ takes on a uniform distribution, and the corresponding upper bound on the entropy can be calculated using the above equation. In general, the lower the entropy, the lower is the information uncertainty associated with the process, and the easier it is to predict future outcomes based on history.

► **Choosing granularities of prediction.** To predict future outcomes of the value of k, past observations can be used at various granularities. Entropy at a given granularity will demonstrate how effective the granularity is in prediction. We consider two orthogonal dimensions to partition past observations:

1. *Collated vs. user-specific*: In a collated model, we assimilate traces of all users to form a collective trace. We then compute one entropy value of the collective trace. In a user-specific model, we consider each user trace in isolation, compute one entropy value for each user, and examine average entropy.
2. *Time independent vs. dependent*: In a time-independent model, we consider all rounds in the history alike and compute entropy from the PDF of T = 1 hour. In a time-dependent model, we maintain 24 PDFs (one for each hour of the day), compute their entropy values, and take the average entropy.[3]

The above two dimensions can produce four combinations of models.

[3] In a time-dependent model, one can consider partitioning a trace even further, such as one PDF for every hour of the week; however, with our limited dataset, such model becomes sparse and useless for prediction.

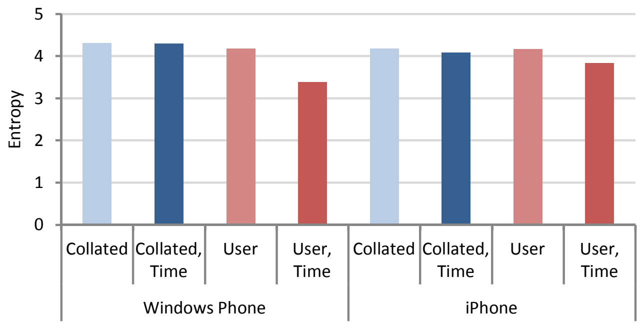


Figure 5. Entropy of app usage in our two datasets, at different granularities.

Figure 5 shows the entropy of the two datasets under various models. The label `Collated,Time` on the x-axis, for example, denotes that we compute a time-dependent model over the collective trace of all users. The results highlight a number of key points. First, entropy is in general high. If we assume that past observations are completely random and hence useless in prediction, the entropy becomes $\log_2 61 \approx 6$. In both datasets and under all models, entropy is closer to this upper bound than the lower bound of 0. This suggests that app usage times in our dataset are mostly unpredictable. Note that this conclusion is based on the assumption that we use past app usage durations, user ID and time of use only. Prediction quality is likely to increase (and hence entropy is likely to decrease) if we use other information such as user's location context, as shown in previous work [38]. On the other hand, the ad server already has access to access logs that have temporal data, but might not have more intrusive data about the user for additional customization.

Figure 5 also shows that considering each user's trace in isolation makes the future outcomes of k a little bit more predictable (as shown by a reduction in entropy for `User`). Finally, considering data from different hours of the day separately further reduces the entropy. The entropy of the user-specific, time-dependent model is lower than all other models. Based on this observation, *we consider each user's trace in isolation and build one model for every hour of the day* in the prediction algorithm that we describe next.

► **Choosing the predictor.** We now consider several statistical predictors to predict how many ad slots will be available in a given round. The goal is to choose the one with the smallest prediction error that we measure with coefficients of variations (root mean square error divided by mean) of the predictions. We consider the following predictors.

- `Sampling` returns a random value sampled from the PDF of the user in the current hour of the day.
- `Avg` returns the average number of slots in the current hour of the day from past observations.
- `k'th percentile` returns the k'th percentile slot count in the current hour of the day from past observations.

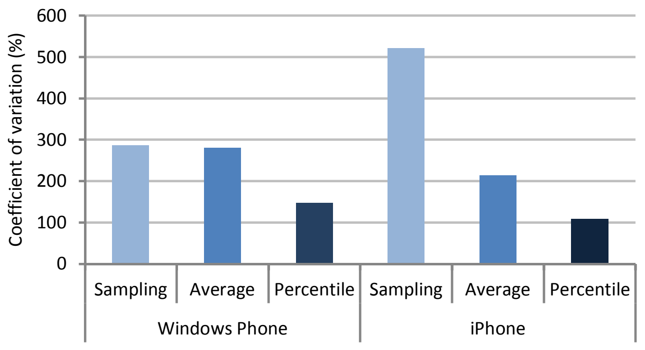


Figure 6. Coefficients of variation (RMSE/mean) of various predictors on user-specific, time-dependent models. We use $k = 50\%$ in the percentile predictor.

Figure 6 shows the result of using these predictors for our two datasets. As shown, the percentile predictor seems to be a good predictor for both the datasets and hence we use it in the rest of the paper. In these tests, we used $k = 50\%$. In the following, we discuss which value of k is best to use.

► **Choosing k for the percentile predictor.** The coefficient of variation metric treats both underprediction and overprediction equally. In practice, they have different effects: underprediction forces the client to prefetch smaller numbers of ads, thus increasing its communication energy cost, while overprediction causes more frequent SLA violations. The best percentile to use depends on the relative importance of energy and SLA violations.

We experimentally evaluate the impact of the percentile predictor on energy efficiency and number of SLA violations in a complete ad prefetching system. Our system works as follows. The client contacts the proxy when it has an ad slot but no ad to display. The proxy uses the percentile predictor to predict how many ads (m) the client might need in the next T time, where T is the *prediction period* smaller than the ad deadline D. It then collects m ads, by prefetching them from the ad exchange or from its pool of previously prefetched ads, and sends them to the client. If the client runs out of ads before time T, it simply contacts the proxy again for additional ads. On the other hand, if the client has displayed only $m' < m$ ads during time T, it returns the undisplayed ads to the server and gets a new batch of ads for the next prediction period. The undisplayed ads (that have now smaller lifetimes) are sent to other clients who have higher probabilities of showing ads. Finally, the proxy also ensures that ads targeted to different profiles/apps are shown only in ad slots of a matching type.

For the evaluation, we use the Windows Phone logs to generate a realistic client workload. Unless otherwise specified, we assume all ads have the same deadline D of 30 minutes, the same type, and the same price. We also conservatively assume that a new ad is shown every minute while the user is using an app; a shorter ad refresh period will improve the relative benefit of prefetching on the battery life-

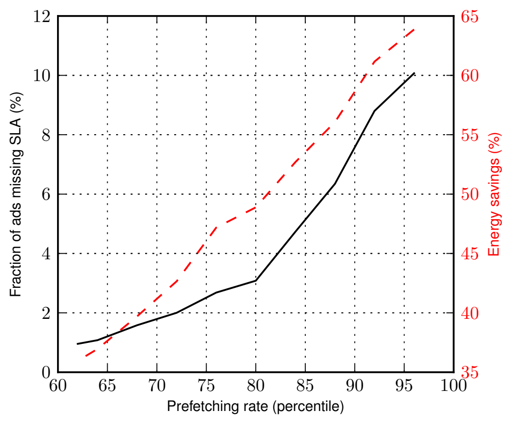


Figure 7. Tradeoff between energy savings and SLA violations for increasingly larger prefetching rates (controlled by k of the kth percentile prediction model). The prediction interval is 20 minutes.

time. We use ads of size 5 kB each, which is the average ad size in the top 15 apps we used in Section 2. To measure energy, we capture network traces from our experiment, feed it into WattsOn used in Section 2, and measure communication energy for a phone using AT&T 3G wireless (the same setup we used in Section 2). We report the energy savings compared to a baseline client that fetches ads one by one, as in today's ad systems.

To determine the best value of k, we plot both average energy savings and SLA violations as a function of k in Figure 7. We use a prediction interval of 20 minutes. As shown, there is an almost linear increase in energy savings until it reaches the point where most of the batches are larger than the set of shown ads (around the 90th percentile). More interestingly, we see that the number of ads whose SLA is violated remains relatively low until the 80th percentile and then shoots up sharply. This suggests that $k = 80\%$ represents a sweet spot between energy and SLA violation. Therefore, we use 80th percentile as our predictor in the rest of the paper.

To conclude, we observe that above we assumed the predictor uses only app usage history of users, in particular distribution of usage durations. It may be possible to improve prediction accuracy by using additional information such as user's context, correlation of usage patterns of various apps, etc. We expect that even though the prediction can get better by using additional such information, as long as there are some prediction errors, ad prefetching will affect SLA and energy efficiency.

On the other hand, there are other ways for the proxy to limit the risk of causing SLA violations. We evaluate them next.

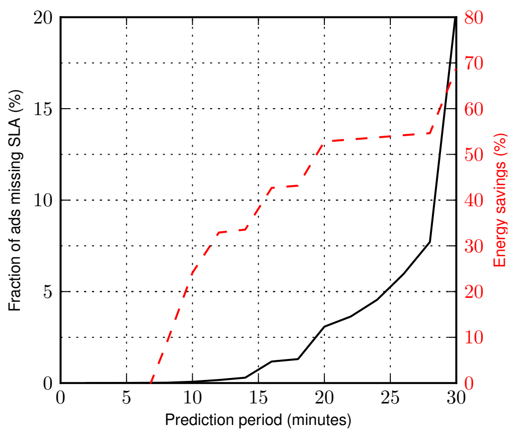


Figure 8. SLA violation rate and reduction in client's energy consumption for increasingly infrequent prediction. The number of ads prefetched is predicted using the 80th percentile prediction model.

4.2 Evaluating tradeoffs

Using the same setup previously described, we evaluate how prediction period, ad deadlines and background traffic can impact on the energy savings and SLA violation rates achieved.

Impact of prediction periods. Increasing the prediction period should intuitively increase the energy efficiency since the client device fetches larger batches of ads. On the other hand, this increases also the period of uncertainty on the status of the ads downloaded by the client. Figure 8 shows the percentage of the ad inventory that incurred an SLA violation and the corresponding reduction in energy consumption for increasingly longer prediction intervals. Since the ads have a deadline of 30 minutes, if the prediction period is longer than or equal to 30 minutes, then it is effectively equivalent to the client not reporting the status of the prefetched ads before their deadline. We see from the graph that for a prediction interval between 15–20 minutes the client achieves a net energy reduction of 40–50% while fewer than 3% of the ads in the inventory experience an SLA violation. To achieve higher savings in energy consumption, a longer prediction interval can be chosen at the cost of increasing the SLA violations. We also observe that after 20 minutes the energy savings are relatively constant for increasing values of the prediction interval. For these reasons, for ads with deadline of 30 minutes, we consider a reasonable prediction interval to be 15 or 20 minutes.

Impact of ad deadlines. Another parameter to consider when trading off energy savings with the number of SLA-violated ads is the ad deadline. Figure 9 illustrates this trade-off. Longer ad deadlines allow for less frequent prediction. For example, the same prediction period of 15 minutes that

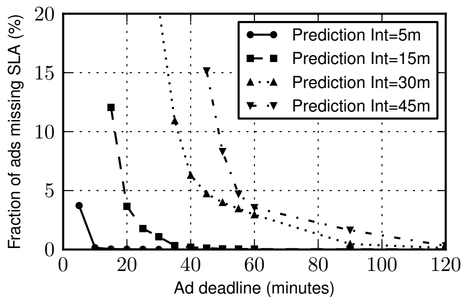


Figure 9. Tradeoff between ad deadlines and prediction period. The longer the ad deadlines, the smaller the client-proxy prediction period required for maintaining the same SLA violation rate. Prefetching uses the 80th percentile prediction model.

we considered above generates almost no SLA violations for ads with an hour or longer deadline. To put this in perspective, our analysis of an existing production ad platform (see Section 2) shows that only 1% of the ads change their bid price in less than an hour, which means that ad prefetching basically incurs no penalty on the ad infrastructure.

Impact of background traffic. A measurement study [37] analyzing the data connections of more than 3 million subscribers of a major European mobile network showed that, in general, ad traffic is likely to occur in isolation from other traffic—no background traffic is present when ad requests are generated. Our results are in accordance with this finding.

We simulate background network traffic with interflow times following a Poisson distribution (memoryless) with differing values of the distribution mean. Our proxy opportunistically uses the presence of background traffic such that if network activity is observed, reports are generated and sent back to the ad server. As the network radio is already active, these reports come at low energy cost. If no network activity is observed during the reporting period, the proxy delays its transfers by a maximum of 2 minutes. This approach allows the proxy to reduce even further the energy overhead of ads. Conversely, in the existing ad architecture, where ads are retrieved on demand, unless a background network flow is initiated in the time interval immediately preceding the ad request (as the network radio is active only for a maximum period of about 17 seconds [25]), no energy saving occurs. Figure 10 shows the amount of ad traffic that is sent when the radio is already active due to the background traffic, for both the prefetching and the baseline (non prefetching) case. In the prefetching case, we use the 80th percentile prediction model, and we assume an ad deadline of 30 minutes and a prediction intervals of 20 minutes. We observe that when the background network flows are on average distributed within 10 minutes of each other, even 78% of the ad network connections occur at low energy cost. On the other hand, the

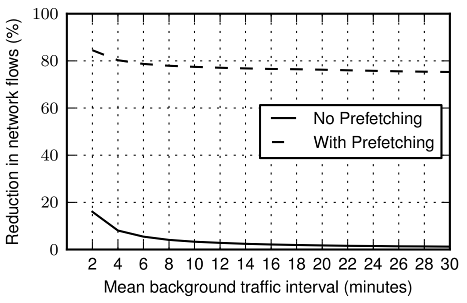


Figure 10. Reduction in the number of network flows that require the radio to be activated when our prefetching model is enabled or disabled. Prefetching uses the 80th percentile prediction model.

non-prefetching system is unable to exploit the background network flows unless they occur extremely frequently.

Overall, based on these experiments, we conclude that for ad deadlines of 30 minutes, using the 80th percentile model with a prediction interval of 20 minutes, we can reduce the energy overhead of ads by as much as 50%, while impacting the SLAs of only 3% of the ad inventory. For ads with deadlines longer than 30 minutes, a prediction interval of 15 minutes is sufficient to eliminate the problem of SLA violations. In the presence of background traffic, the energy reduction is even higher.

5. Overbooking model

The app usage prediction model guarantees a negligible number of SLA violations for ads with deadlines over 30 minutes, but could the proxy deal with shorter deadlines?

We explore whether advertising systems can take advantage of research in the area of overbooking of temporal resources [10, 33], which besides the traditional use cases of airline and hotel reservation systems, has been shown to be effective also for resource provisioning in the cloud [36].

To support overbooking, we modify our prefetching system as follows: (a) The proxy maintains a queue of unexpired, pending ads that have already been sent to some clients. (b) Each time a new client request is received, the proxy computes not only an estimation of how many ad slots the client will have in the next prediction interval, but also the probability of each of those slots being used. This can be computed from the PDF of historical slot counts of the user. (c) On a request of new ads, the proxy sends to the client not only a set of ads, but also the information about which ad to be shown in which slot. (d) The proxy can send new ads to a client, or *overbook* (or replicate) some of the pending ads.

Intuitively, overbooking or sending an ad to multiple clients increases the chance that the ad will be displayed by at least one client and hence decreases the SLA violation rate. However, it entails the risk of displaying the same ad in multiple client slots while only being paid for one impression by the advertiser (i.e., revenue reduction may occur). The goal of the overbooking model is to maximize the number of distinct ads that can be shown given a certain number of client ad slots. In particular, prefetched ads which are unlikely to be shown, and only those, should be replicated across clients more aggressively. For the next experiments, we conservatively assume that there is no background network traffic for opportunistic notifications to the proxy.

Overbooking algorithm. Each time a client device requests a set of ads, the overbooking model attributes a *showing probability* to each of its pending ads. For a given pending ad, let S denote the set of ad slots (in different clients) it has been sent to, and let $P(S_i)$ denote the probability of the ith slot in S being used. Let X be the random variable denoting the number of times the pending ad will be displayed. Then,

$$\begin{aligned}
P(X \geq 1) &= 1 - \prod_i (1 - P(S_i)) \\
P(X = 0) &= 1 - P(X \geq 1) \\
P(X = 1) &= P(S_1) \prod_{i \neq 1} (1 - P(S_i)) + \dots \\
&\qquad P(S_n) \prod_{i \neq n} (1 - P(S_i)) \\
P(X > 1) &= P(X \geq 1) - P(X = 1)
\end{aligned}$$

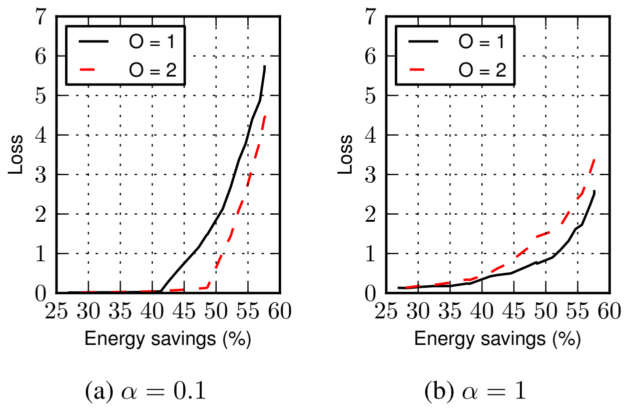
$P(X = 0)$ is the probability that an SLA miss will occur for the ad and $P(X \geq 1)$ is the probability that multiple displays will be made.

Each time a request is made for a batch of ads, the proxy iterates through the set of ads it has already retrieved from the ad exchange whose display status is unknown, and verifies if the penalty for associating the ad with a given slot will increase or decrease. If the penalty decreases, the ad is associated to the slot that most minimizes its penalty. The penalty function is defined as:

$$Penalty = P(X \geq 1) \times O + P(X = 0)$$

The parameter O is the *overbooking threshold* value that the proxy uses to tune the aggressiveness of the overbooking model. The smaller the value of O, the more aggressive overbooking is. We use the same value of O for all ads, but this could also be used to prioritize certain types of ads over others, potentially based on revenue.

The above penalty function can be computed for ads that have already been sent to other clients (and hence the set S of slots they are attached to is non empty). For an ad which is currently not sent to any client, however, this is not true. For such an ad, we use the following procedure to pick a slot. Each such ad has a lifetime d, computed as the difference between its original deadline D and the time elapsed since it was first prefetched. We would like to attach shortly expiring ads to the first ad slots available, with higher probability of being used. We therefore assign the ad to any of the first



(a) $\alpha = 0.1$ (b) $\alpha = 1$

Figure 11. Effect of different overbooking thresholds.

$d/D \times B$ slots, where B is the batch size predicted. For example, if an ad is not attached to any slot (in any client) yet, but its lifetime is only 1/3rd of its deadline, we assign it to any of the first 1/3rd slots. This ensures that the shorter the lifetime, the more aggressively the proxy puts the ad on slots with higher showing probability.

Evaluating overbooking. Overbooking results in decreased SLA violations at the cost of revenue loss. Different ad networks may have different preferences towards these two factors. For example, in existing ad systems, when the ad proxy retrieves an ad from the ad exchange, the ad network who won the auction assumes that it will be immediately displayed on the client. This is important because the ad network has to manage its own campaigns from different advertisers. Thus, it is expected that the proxy will try to reduce the SLA violations as much as possible, even at the cost of losing some revenue (when ads are displayed multiple times). This preference can be controlled by the overbooking threshold O.

Figure 11 shows the effects of two different overbooking thresholds. We unify SLA violation and revenue loss into a single loss metric: $(\alpha \times sla\ violations + (1-\alpha) \times rev\ loss)$, which allows ad networks to weigh the two factors according to their preference. We report the loss metric for two values of α. As shown, when SLA violation is more important than revenue loss (i.e., $\alpha = 1$), the ad network should use aggressive overbooking, with a smaller value of the overbooking threshold (e.g., $O = 1$). On the other hand, when revenue is more important, the ad network should use conservative overbooking, with a larger value of O.

6. Related work

Previous work looked at the energy cost of mobile ads and proposed prefetching and caching to limit the overhead. Vallina-Rodriguez et al. [37] provide an in-depth characterization of mobile ad traffic using traces of over 3 million subscribers of a major European mobile network. They propose a phone prototype for prefetching and caching ads and show the energy improvements achievable. The authors of [17] focus their study on the ad overhead of 13 popular Android apps and sketch the proposal for a middleware that prefetches ads when network conditions are most favorable. No implementation or evaluation is provided. Finally, a third study by Pathak et al. [28] evaluates the high network cost of advertising using a fine-grained energy profiler for smartphones. They test the tool with 6 popular Android apps and find that ads in a game such as Angry Birds consume 45% of the total energy. We share the same goal of the first two studies, but we focus on designing and evaluating a solution compatible with the current ad ecosystem, particularly with the real-time bidding model of ad exchange. Our energy measurements are in accordance with the findings of all these three studies. We further isolate the communication overhead of advertising from that of the app itself in order to get a more precise estimate.

As discussed in Section 2.4, the high communication energy costs of mobile ads are due to the 'tail time' of 3G networks. A long tail time improves responsiveness, but causes a large energy tail. Balasubramanian et al. [7] show that in 3G networks, about 60% of the energy consumed for a 50 kB download is tail energy. In a GSM network where the tail time is shorter, network transfers consume 40% to 70% less energy compared to 3G, but suffer from higher network latency. WiFi is not affected by the tail energy problem, but incurs a high initialization cost for associating with an access point. Solutions to the tail energy problem looked into determining the optimal tail time for different 3G networks [9, 39]. However, as the tail time is under the carrier's control, they require changes at the network operator's side. Other solutions use "cheaper" radios to reduce the wake up and tail energy of the cellular radio [7]. For instance, Cell2Notify[2] uses the GSM radio to wake up WiFi. Others, such as Coolspots [29], switch from low-power/low-bandwidth interfaces (Bluetooth) to high-power/high-bandwidth interfaces (WiFi), when an application requires it. In general, these approaches are orthogonal to the savings that prefetching ads in bulk provides. Finally, application-specific solutions have been considered as well. Balasubramanian et al. [7], for example, propose efficient scheduling of network transfers for delay tolerant applications (email, news feeds and software updates) and prefetching for web search and browsing. Our solution falls in this category and focuses on the dynamics of modern advertising networks.

In general, prefetching techniques have been studied extensively in the context of web browsing, both for desktops [16, 23, 26, 27] and mobile devices. For mobile devices, prefetching has been used to support disconnected operation [18, 40], or to reduce access latency and power consumption [4, 7, 40]. Predicting a user's web accesses typically relies on determining the probability of the user accessing web content based on previously accessed content, by extracting sequential and set patterns [14, 26], as well as temporal features for higher prefetching accuracy [22]. Our app usage prediction model is based only on temporal access patterns. The main difference between web content prefetching techniques and ad prefetching is that for web content the prefetching strategy is entirely client-driven, whereas for ads the server needs to be in control to minimize the risk of revenue loss.

Techniques to preemptively load mobile apps for reduced launch time use location features to predict which apps are more likely to be used in specific locations [38]. Our system could use these techniques to improve targeting of ads.

Finally, ad systems have been the focus of several recent projects [5, 13, 35] mainly because of the privacy issues they raise. Privacy-preserving architectures such as Adnostic [35] and Privad [13] allow users to disclose only coarse-grained information about themselves, while personal profiles are kept local and use to rank relevant ads. Although the main motivation of our work is not privacy, interestingly, these architectures build on the assumption that prefetching ads in bulk is a feature supported by ad servers. For instance, Privad uses subscription-based prefetching where the user manually subscribes to ads category of interest and corresponding ads are fetched at a proxy whenever available. Our solution to ad prefetching, which is also compatible with the existing ad infrastructure, can make privacy-preserving ad system viable.

7. Conclusion

Although content prefetching is a well-established practice in several domains (file systems, web browsing and search), applying it to ad systems is non trivial. The on-demand nature of the ad exchange's auction model does not lend itself easily for disconnected operation. The advantage of prefetching ads in bulk for smartphone users is significant: we show the communication energy consumed by ads can be reduced by 50% or more. The incentive for the ad ecosystem to support such a model is manifest by the largest mobile advertising platforms being owned by the largest smartphone providers. The remaining question is: can ad platforms afford to support such a model? This paper explores this question by architecting an ad prefetching system that relies on app usage prediction models to estimate how many ads users are likely to consume in the future, and an overbooking model to probabilistically replicate ads across clients.

We show the tradeoffs involved in tuning such a system to provide energy savings with limited revenue loss for the ad system. Specifically, the risk of not being able to show all prefetched ads by their deadlines and unnecessarily showing the same ad multiple times should be kept to a minimum. This paper is a first step towards establishing an ad prefetching model on which more complex functionality can be built. Among those, privacy is one we intend to explore in the future. Recent systems have already shown ad prefetching to be essential in supporting anonymization of users' personal information, but it is unclear whether the prefetching bandwidth is sufficient to achieve solid privacy guarantees.

8. Acknowledgments

We thank the reviewers for their insightful comments, and especially our shepherd, Steven Hand, for helping us improve the paper and its presentation. We thank the members of the LiveLab project at Rice University for providing us with the iPhone dataset. Finally, thanks to Asela Gunawardana for helping us with the app usage prediction model.

References

[1] Flurry. http://www.flurry.com.

[2] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta. Wireless wakeups revisited: energy management for voip over wi-fi smartphones. In *Proceedings of the 5th international conference on Mobile systems, applications and services (MobiSys '07)*, pages 179–191. ACM, 2007.

[3] App Brain. Free vs. paid android apps. http://www.appbrain.com/stats/android-app-downloads.

[4] T. Armstrong, O. Trescases, C. Amza, and E. de Lara. Efficient and transparent dynamic content updates for mobile clients. In *Proceedings of the 4th international conference on Mobile systems, applications and services (MobiSys '06)*, pages 56–68. ACM, 2006.

[5] M. Backes, A. Kate, M. Maffei, and K. Pecina. ObliviAd: Provably Secure and Practical Online Behavioral Advertising. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pages 257–271, may 2012.

[6] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting Mobile 3G Using WiFi. In *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10)*, pages 209–222. ACM, 2010.

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

[8] BlackBox Research. Smartphones in singapore: A whitepaper release. http://www.blackbox.com.sg/, May 2012.

[9] M. Chuah, W. Luo, and X. Zhang. Impacts of inactivity timer values on UMTS system capacity. In *Wireless Communications and Networking Conference (WCNC '02)*, volume 2, pages 897–903, mar 2002.

[10] P. Davis. Airline ties profitability to yield management. *SIAM News*, 1994.

[11] DngNgo. Survey: Wi-Fi becoming smartphone must-have. http://news.cnet.com/8301-17938_105-10209682-1.html, 2009.

[12] Google Inc. Google AdMob Ads SDK. https://developers.google.com/mobile-ads-sdk/docs/admob/intermediate.

[13] S. Guha, B. Cheng, and P. Francis. Privad: Practical Privacy in Online Advertising. In *Proceedings of the 8th Symposium on*

Networked Systems Design and Implementation (NSDI '11), pages 13–13, Mar 2011.

[14] Z. Jiang and L. Kleinrock. Web prefetching in a mobile environment. *IEEE Personal Communications*, 5(5), 1998.

[15] J.-M. Kang, S. seok Seo, and J. W.-K. Hong. Usage pattern analysis of smartphones. In *Proceedings of 13th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 21-23 Sept 2011.

[16] F. Khalil, J. Li, and H. Wang. Integrating recommendation models for improved web page prediction accuracy. In *Proceedings of the thirty-first Australasian conference on Computer science (ACSC '08)*, pages 91–100. Australian Computer Society, Inc., 2008.

[17] A. J. Khan, V. Subbaraju, A. Misra, and S. Seshan. Mitigating the true cost of advertisement-supported "free" mobile applications. In *Proceedings of the 12th Workshop on Mobile Computing Systems & Applications (HotMobile '12)*, pages 1:1–1:6. ACM, 2012.

[18] A. Komninos and M. Dunlop. A calendar based internet content pre-caching agent for small computing devices. *Journal of Personal and Ubiquitous Computing*, 12(7), 2008.

[19] E. Koukoumidis, D. Lymberopoulos, K. Strauss, J. Liu, and D. Burger. Pocket cloudlets. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS "11)*, pages 171–184. ACM, 2011.

[20] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo. Don't kill my ads!: balancing privacy in an ad-supported mobile application market. In *Procedings of the 12th Workshop on Mobile Computing Systems and Applications (HotMobile '12)*, page 2. ACM, 2012.

[21] LiveLab traces. Rice university. http://livelab.recg.rice.edu/traces.html.

[22] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas. PocketWeb: instant web browsing for mobile devices. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '12)*, pages 1–12. ACM, 2012.

[23] E. P. Markatos and C. E. Chronaki. A top-10 approach to prefetching on the web. In *Proceedings of Internt Society Conference (INET)*, 1998.

[24] Microsoft Research. Common Compiler Infrastructure (CCI). http://research.microsoft.com/en-us/projects/cci/.

[25] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (MOBICOM '12)*. ACM, August 2012.

[26] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized web prefetching. *IEEE Transactions on Knowledge and Data Engineering*, 15:1155–1169, Sept-Oct 2003.

[27] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *SIGCOMM Comput. Commun. Rev.*, 26(3):22–36, July 1996.

[28] A. Pathak, Y. C. Hu, and M. Zhang. 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 (Eurosys '12)*, pages 29–42. ACM, 2012.

[29] T. Pering, Y. Agarwal, R. Gupta, and R. Want. CoolSpots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Proceedings of the 4th international conference on Mobile systems, applications and services (MobiSys '06)*, pages 220–232. ACM, 2006.

[30] A. Rahmati and L. Zhong. Context-for-wireless: context-sensitive energy-efficient wireless data transfer. In *Proceedings of the 5th international conference on Mobile systems, applications and services (MobiSys '07)*, pages 165–178. ACM, 2007.

[31] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum. LiveLab: measuring wireless networks and smartphone users in the field. In *ACM SIGMETRICS Perform. Eval. Rev*, volume 38. ACM, December 2010.

[32] C. Shin, J.-H. Hong, and A. Dey. Understanding and Prediction of Mobile Application Usage for Smart Phones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (UbiComp '12)*, pages 173–182. ACM, 2012.

[33] B. Smith, J. Leimkuhler., and R. Darrow. Yield management at american airlines. *Interfaces*, 22(1):8–31, 1992.

[34] W. L. Tan, F. Lam, and W. C. Lau. An empirical study on the capacity and performance of 3g networks. *IEEE Transactions on Mobile Computing*, 7:737–750, June 2008.

[35] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy Preserving Targeted Advertising. In *Proceedings of the Network and Distributed System Security Symposium (NDSS '10)*, 2010.

[36] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI '02)*, December 9-11 2002.

[37] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. Breaking for commercials: characterizing mobile advertising. pages 343–356. ACM, November 14-16 2012.

[38] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th international conference on Mobile systems, applications, and services (MobiSys '12)*, pages 113–126. ACM, 2012.

[39] J.-H. Yeh, J.-C. Chen, and C.-C. Lee. Comparative Analysis of Energy-Saving Techniques in 3GPP and 3GPP2 Systems. *IEEE Transactions on Vehicular Technology*, 58(1):432 –448, jan. 2009.

[40] L. Yin and G. Cao. Adaptive power-aware prefetch in wireless networks. *IEEE Transactions on Wireless Communications*, 3:1648–1658, Sept 2004.