Energy-Aware Advertising through Quality-Aware Prefetching on Smartphones

Yi Yang, Yeli Geng and Guohong Cao
Department of Computer Science and Engineering
The Pennsylvania State University
Email: {yzy123, yzg5086, gcao}@cse.psu.edu

***Abstract*—In-app advertising provides a monetization solution for free apps, but also consumes lots of energy due to the long tail problem in cellular networks. To reduce the tail energy, we can predict the number of ads needed in the future and then prefetch those ads together instead of periodically. However, prefetching unnecessary ads may waste both energy and cellular bandwidth, and this problem becomes worse when the network quality is poor. In this paper, we propose network quality aware prefetching algorithms. First, we design a prediction algorithm which generates a set of prefetching options with various probabilities. Second, with these prefetching options, we propose two prefetching algorithms to reduce the energy consumption by considering the effect of network quality, where the energy-aware prefetching algorithm aims to minimize the energy consumption, and the energy-and-data aware prefetching algorithm considers the data usage to achieve a tradeoff between energy and data usage. Evaluation results show that, compared to traditional ways of fetching ads periodically, our energy-aware prefetching algorithm can save 80% of energy under various network quality, and our energy-and-data aware prefetching algorithm can achieve similar energy saving with less data usage.**

I. Introduction

With the proliferation of smartphones, people spend a large amount of time on mobile apps. There are two kinds of mobile apps: free apps, and paid apps. According to recent studies, free apps account for above 90% of the total amount of apps downloaded in the market in 2015 [1], [2]. To pay for the cost of the app development, free apps are usually associated with in-app advertising [3].

Although users enjoy using free apps, the energy consumption of fetching in-app ads through the cellular network may lead to significant battery drain on smartphones. For example, a recent measurement study based on top 15 ad-supported windows phone apps shows that fetching in-app ads consumes 23% of the app's total energy and 65% of the app's total communication energy [4].

In cellular networks, the release of radio resource is controlled by multiple timers, and the timeout value can be more than 10 seconds [5], [6], [7], [8]. Thus, it is possible that the cellular interface continues to consume a large amount of energy (also referred to as the *long tail problem*) before the timer expires, even when there is no network traffic. Although it only takes less than one second for the cellular interface to fetch an ad, due to the long tail problem, much more energy may be wasted. This problem becomes worse since ads are fetched periodically, and then the long tail problem occurs frequently [9]. To reduce the energy related to the long tail problem, techniques such as fast dormancy [10] have been proposed. Fast dormancy can reduce the tail time by switching the cellular interface into the low power state immediately after the data transmission. However, this requires support from both mobile devices and cellular carriers. Furthermore, it may not know when the next data transmission will happen. If the next data transmission happens quickly, fast dormancy may waste energy and introduce extra delay on switching the smartphone out of the low power state.

As another solution to address the long tail problem, we can predict the number of ads needed in the future and then prefetch those ads together. Then, only one tail is generated instead of multiple tails with the traditional ways to fetch ads periodically. Although prefetching multiple ads can reduce the tail energy, its potential cost is also high. This is because prediction may be inaccurate, and prefetching unnecessary ads may waste both energy and cellular bandwidth. This problem becomes worse when the network quality is poor and it will take much longer time to transmit the same amount of data and consume more energy. Thus, prefetching should also be aware of the network condition.

Since it is hard to know the exact number of ads to prefetch, which is app-dependent and user-dependent, we have to adjust the number of ads to prefetch based on the network quality. Generally speaking, more ads should be prefetched when the network quality is good, and fewer ads should be prefetched when the network quality is poor. Although redundant ads may be prefetched under good network quality, it avoids other possible long tail problems. Similarly, when the network quality is poor, prefetching fewer ads can avoid the energy waste of prefetching unneeded ads.

In this paper, we propose network quality aware prefetching algorithms. Different from traditional data-mining based prediction algorithms which only generate one option (i.e., the number of ads to prefetch), the proposed prediction algorithm generates a set of options with various probabilities. With these prefetching options, we propose two prefetching algorithms to reduce the energy consumption by considering the effect of network quality, where the energy-aware prefetching algorithm aims to minimize the energy consumption, and the energy-and-

This work was supported in part by the National Science Foundation (NSF) under grants CNS-1421578 and CNS-1526425.

data aware prefetching algorithm also considers the data usage to achieve a tradeoff between energy and data usage.

The contributions of this paper include:

- We propose a prediction algorithm to generate a set of prefetching options with various probabilities, so that different options are adopted based on the network quality to save energy.
- With multiple prefetching options, we propose two prefetching algorithms: the energy-aware prefetching algorithm aims to minimize the energy consumption, and the energy-and-data aware prefetching algorithm achieves a tradeoff between energy and data usage.
- We have implemented and evaluated the proposed prefetching algorithms under different network quality. Evaluation results show that, compared to traditional ways of fetching ads periodically, our energy-aware prefetching algorithm can save 80% of energy, and our energy-and-data aware prefetching algorithm can achieve a similar energy saving with less data usage.

The rest of this paper is organized as follows. Section II discusses related work. Section III presents some preliminaries. Section IV introduces network quality aware prefetching algorithms. We evaluate the proposed algorithms in Section V and test them in a real app in Section VI. Section VII concludes the paper.

II. Related Work

Recently, lots of research has been done on energy consumption of fetching in-app ads. A detailed measurement about the ad volume has shown that 50% of Android users have spent more than 5% of their total network traffic related to ad [11]. Although it seems not very large, fetching these ads will generate periodical data transmissions, which contribute to 30% of the total radio energy consumption due to the long tail problem [9]. A case study in [12] has found that 70% of the energy consumed in Angry Birds (a game app) on Android devices is related to a third party ad library.

Prefetching has been adopted to solve the long tail problem in cellular networks [13], [14]. Parate *et al*. [13] proposed to predict what app will be used next, and then prefetch that app. Although they solve the problem of determining what data to prefetch, they do not consider the problem of how long an app will be used and how much data (ads) to prefetch, which is the focus of this paper. In [14], a system was built to support informed prefetch in which developers can use an API to provide a hint for prefetch. However, in practice it is hard for developers to know how much data will be needed in the future, which is user-dependent (e.g., how many ads are needed depends on how long an app is used). Some prefetching algorithms [15], [16] prefetch all the data when a fast network connection (e.g., WiFi, LTE with good network quality) is available. However, in reality it is hard to know if such a fast network connection will be available before the data is needed (e.g., An indoor area with poor network quality and no WiFi). In our work, we do not have this assumption, and we try to find the most energy efficient way for prefetching under the current network quality.

Our work is mostly related to [4], [17] which attempt to reduce the energy consumption of fetching in-app ads. The CAMEO framework [17] provides a middleware to prefetch context-dependant ads for all apps. It predicts what apps will be used in the future to determine ad contexts and prefetches a fixed number of ads for each app. However, the number of ads to prefetch is app-dependent and user-dependent, which is not fixed. Our work can be considered complimentary to CAMEO, as our work focuses on predicting the number of ads to prefetch for certain app and user under the current network quality. Mohan *et al*. [4] proposed an overbooking model at the ad server/network to ensure that all ads can be displayed before deadline, and predicted how many ads to prefetch by using the 80th percentile value of the number of ads displayed in historical records. Different from using a simple rule to predict the number of ads to prefetch, which may consume much more energy under some network quality, our prefetching algorithms determine the number of ads to prefetch by considering the effect of network quality.

III. Preliminaries

In this section, we first provide some background of how in-app advertising works, and then present our design considerations and basic ideas.

A. Background: In-app Advertising

We first briefly illustrate the current in-app ad ecosystem, and then introduce the in-app ad format and size. At the end, we discuss what ads to prefetch.

1) In-app Ad Ecosystem: Current in-app advertising ecosystem involves three main parts: Apps, Ad Networks, and Advertisers. Apps on the smartphones rely on the embedded ad library, which is usually provided by the same company who also provides the corresponding ad network, to fetch and display ads. Advertisers can register with the ad network and initiate ad campaigns. An ad campaign is a contract between advertisers and the ad network (e.g., delivering 10,000 ads within a day). The responsibility of the ad network is to complete all the ad campaigns. Current ad networks usually deploy real-time bidding (RTB) strategies [18] to display the most valuable and relevant ads for mobile users. Although the bidding price of an ad may change in a RTB-based system, recent studies [4] have shown that the bidding price is stable in a short period of time (e.g., several hours), which is longer than most app running time.

2) In-app Ad Format and Size: In-app ad formats include banners, rich media, and video. According to recent market analysis and prediction [19], rich media is becoming the most popular ad format. Thus, we consider to prefetch rich media ads in this paper. When prefetching a rich media ad, the whole file of the ad needs to be fetched to avoid extra network activities, although some parts of the file are only useful after certain user interactions. According to ad specifications [20], [21], [22], [23], the whole file size of a rich media ad ranges from 50 KB to 200 KB.

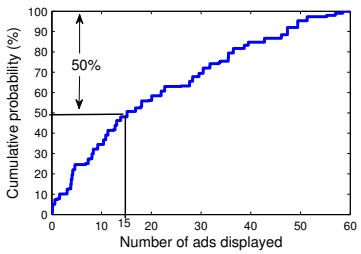


Fig. 1. Cumulative distribution of how many ads are displayed

3) What Ads to Prefetch: Ads to be displayed are determined by app-dependent contexts such as app category [17], and then it is easier to decide what ads to prefetch for a specific app. This will be much harder for all apps because different apps may have different contexts and then different types of ads should be prefetched. Thus, ads are only prefetched for a certain app in this paper, and we know what app is under consideration and what ads to prefetch. Specifically, ads are prefetched when new ads are needed in the app, and unused ads are discarded when the app is closed.

B. Design Considerations and Basic Ideas

The energy consumption of prefetching ads is affected by the prediction accuracy. If the prefetched ads are more than necessary, prefetching those unneeded ads may waste a lot of energy. On the other hand, if the prefetched ads are fewer than needed, more prefetches will be needed, and resulting in extra long tail problems. In cellular networks, the energy consumption of prefetching ads varies with the network quality. For example, as we measured in a LTE network, when the network quality is poor, downloading a 100 KB ad consumes 2 joules. When the network quality is good, downloading a 100 KB ad consumes 0.1 joules. The energy consumption of a long tail remains about 10 joules under different network quality.

Traditional data-mining based algorithms do not consider the network quality, and only generate one option (i.e., the number of ads to prefetch), which may be too high or too low and then wasting some energy. To further reduce the energy consumption, we have to consider the network quality by designing adaptive algorithms which can adjust the number of ads to prefetch accordingly. To achieve this, we generate multiple options based on the historical app usages, and choose the one with the least energy consumption according to the current network quality.

For example, Fig. 1 shows the cumulative distribution of the number of ads displayed for an app according to our trace. The largest number of ads displayed is 60 and the median is 15. We can prefetch 60 ads, or we can prefetch 15 ads with a 50% probability that 45 more ads are prefetched in the future. Under poor network quality (i.e., downloading an ad consumes 2 joules), prefetching 60 ads consumes $2 \times 60 + 10 = 130$ joules. As the second option, we can prefetch 15 ads, and then the expected energy consumption is $2 \times 15 + 10 + 50\% \times (2 \times 45 + 10) = 90$ joules. In this case, prefetching fewer ads (the second option) wins. Under good network quality (i.e., downloading an ad consumes 0.1 joules), prefetching 60 ads consumes $0.1 \times 60 + 10 = 16$ joules. As the second option, we can prefetch 15 ads, and then the expected energy consumption is $0.1 \times 15 + 10 + 50\% \times (0.1 \times 45 + 10) = 18.75$ joules. In this case, prefetching more ads (the first option) wins.

The above example shows the basic idea of our network quality aware prefetching algorithms. We first design a prediction algorithm which generates multiple prefetching options (i.e., the number of ads to prefetch) with detailed information to estimate the probability of future prefetches. Then, we estimate the energy consumption of each option by considering the effect of network quality and choose the best one accordingly.

IV. Network Quality Aware Prefetching

In this section, we first present the prediction algorithm which can generate multiple prefetching options with various probabilities, and then describe two prefetching algorithms to determine the number of ads to prefetch according to the network quality.

A. Prediction Based on a Series of Probabilities

Our prediction algorithm aims to generate multiple prefetching options of how many ads to prefetch. Specifically, our goal is to generate a set of options in the form of (α, n), where α is a predefined probability, and n is the predicted number of ads corresponding to α such that the probability of displaying more than n ads in an app is α. Since the number of ads can be calculated by dividing the app duration, which is from the app being opened to being closed, by the ad refresh interval, which is fixed and easy to find, the goal becomes to predict the app duration corresponding to α.

For a certain app, the prediction can be based on the percentile app duration which can be obtained from the app usage records. For example, the 20th percentile app duration is the time value t below which 20% of the observed app duration can be found. That is, the probability of using the app longer than t is 80%, and the corresponding prefetching option is $(\alpha = 80\%, n = \frac{t}{ad-refresh-interval})$. Thus, to predict the app duration for a certain value of α, we only need to calculate the corresponding percentile app duration.

It may not be a good idea to consider all app usage records for prediction, because some of them may be misleading at the current time and location. For a certain app, a user may use it in some specific context, which has different app duration from other contexts. For example, a student may read newspaper at school during class break or at home. Due to the time limitation in the class break, the student may spend much less time when reading newspaper at school than at home. As a result, those app usage records generated at school are not suitable for predicting the app duration at home, and vise versa. Thus, to predict the app duration, we should only consider those app usage records generated in a similar context. Then, we need to identify the context of the app, and partition app usage records based on the context.

It is very difficult to partition app usage based on context due to two reasons. First, it is hard to know whether a user

TABLE I
MUTUAL INFORMATION BETWEEN APP DURATION AND FEATURES

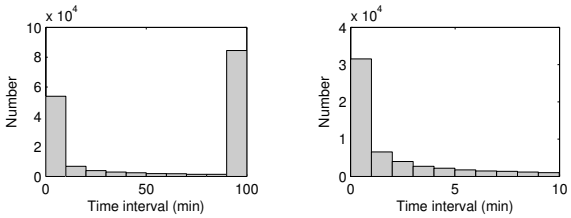
total # of app usage records	170K
total # of apps	2K
H(app duration)	5.59 bits
I(app duration, time)	1.18 bits
I(app duration, location)	0.71 bits
I(app duration, app category)	0.13 bits
I(app duration, last app name)	0.08 bits
I(app duration, last app duration)	0.04 bits
I(app duration, recent call-SMS)	0.02 bits

runs an app in some specific context or just runs it randomly. Second, even if such contexts exist, it is hard to identify them because of the diversity of user behaviors. For example, a user may run an app at particular locations, while uses another app at particular time and locations. Thus, there is no simple rule to identify these contexts.

To address this problem, we adopt the clustering technique, which can group together app usage records in such a way that app usage records in the same group (called a *cluster*) have more similar features (e.g., time and location) than those in other groups. If these app usage records are found tightly grouped (clustered), it means that this app is used in specific contexts, which can be represented by the centroid of clusters. Then, we can classify the current app usage to a cluster, and only use app usage records in that cluster for prediction.

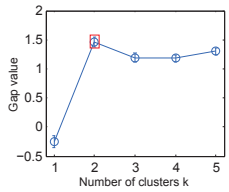
1) Feature Selection: It is important to carefully select features that affect the app duration. In information theory, for a discrete random variable X, the entropy ($H(X)$) measures the uncertainty of X. The mutual information between random variable X and Y, denoted as $I(X, Y)$, represents the mutual dependence between X and Y. A higher mutual information between X and Y suggests that X and Y are more relevant to each other. Thus, features that have the highest mutual information with the app duration should be selected. In our experiment based on the LiveLab dataset [24], we discretize the app duration into 50 values, and calculate the mutual information between it and different features. From the result shown in Table I, time and location have the highest mutual information with the app duration, so they are chosen for clustering.

2) App Usage Record: An app usage record includes information about app duration, time, and location. An interesting observation from the LiveLab dataset is that most of the time interval between two consecutive usages of the same app is either very short (less than 5 minutes) or very long (more than 100 minutes), as shown in Fig. 2, where the time interval is counted as 100 minutes if it exceeds 100 minutes. By analyzing user behaviors within those short time intervals, we find that most app usages are interrupted because the user needs to reply a message or quickly check some information such as clock or weather. Thus, those app usages whose time interval is less than 5 minutes are considered as one usage. Based on this observation, we do not discard unused ads immediately when the app is closed. Instead, five more minutes are waited before discarding those ads. If the app is reused within 5 minutes, it is considered as the same usage as the last one, and those ads can still be used for display.

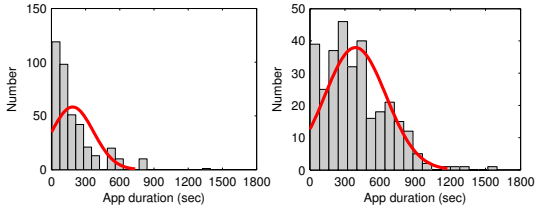


(a) Distribution of time interval (b) Distribution of time interval less than 10 minutes

Fig. 2. Time interval between two consecutive usages of the same app



(a) Gap statistic to determine the number of clusters



(b) App duration distributions in two clusters

Fig. 3. Partitioning app usage records into clusters by K-means

As mentioned above, time and location are selected as features for clustering. In an app usage record, time has two different scales: hour of the day, and day of the week. Location is represented by the cell id. Although cell id based localization incurs errors as high as 500 meters, it is sufficient to recognize coarse-grain locations like “home” or “office”. Unlike GPS based localization, cell id based approach is more energy efficient.

3) Partitioning App Usage Records into Clusters: In order to group together app usage records, different kinds of clustering algorithms can be applied. We choose the most commonly used clustering algorithm, K-means, because of its simplicity. K-means algorithm can partition app usage records into k clusters in such a way that every app usage record is partitioned into a cluster with the nearest mean to the centroid of that cluster. k is assumed to be less than 5, otherwise the app usage records in each cluster may be insufficient to calculate the percentile app duration for prediction. To determine the value of k, we use the gap statistic [25], which is one of the best cluster validity methods for determining the number of clusters for an unsupervised clustering algorithm like K-means. For example, in Fig. 3a, the gap statistic shows a peak at $k = 2$, which means that app usage records can be most tightly grouped into two clusters. After partitioning app usage records into two clusters (C_1 and C_2), we can see that there is a notable difference between the distributions of app duration in these two clusters, as shown in Fig. 3b. We update clusters after every ten app usage records are added. As measured

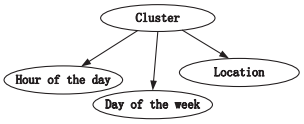


Fig. 4. Naive Bayes classifier

TABLE II
CONFUSION MATRIX

	C_1	C_2
C_1	198	0
C_2	0	100

on a Samsung Galaxy S6 phone, running clustering algorithm on average takes 86 ms and consumes less than 0.1 joules, which is negligible compared to the energy consumption of downloading ads.

4) Classifying Current App Usage to a Cluster: We use the naive Bayes classifier [26] to classify the current app usage to a identified cluster, and then use app usage records in that cluster for prediction. The structure of the naive Bayes classifier is shown in Fig. 4. To evaluate its accuracy, we randomly select some app usage records which have been assigned to a cluster, and then test whether they can be classified to the right cluster. The classification result is represented by a confusion matrix. As an example shown in Table II, our classifier successfully classifies app usage records in cluster C_1 for 198 times and those in cluster C_2 for 100 times without any error. The average accuracy of classification for all clusters is 99.8%.

5) Generating Prediction Results for a Cluster: After classifying the current app usage to a cluster, we use app usage records in that cluster to predict the app duration. First, we define a series of probabilities $\{\alpha_i\}_{i=1}^{M}$, where $100\% > \alpha_1 > \alpha_2 > ... > \alpha_M = 0\%$. Then for each α_i, the app duration is predicted as the corresponding percentile app duration A_i observed in that cluster such that the probability of running the app for more than A_i seconds in the future is α_i, where A_M is the longest app duration observed in that cluster. Finally, we divide A_i by the ad refresh interval, which is a fixed value, to obtain the number of ads n_i to prefetch. The final set of prefetching options is $\mathbb{R} = \{(\alpha_i, n_i) \mid i = 1, 2, ..., M\}$, where n_M is the largest number of ads displayed in history. In this paper, we set $M = 10$ and α_i = 90%, 80%, ..., 10%, 0% to generate prefetching options.

B. Prefetching Algorithms

To reduce the energy consumption of prefetching, network quality is taken into account to determine how many ads to prefetch among options provided by $\mathbb{R}$. In this section, we first provide an overview of the energy model of the LTE network, and then discuss how to define and obtain the network quality. Finally, we give a detailed description of two prefetching algorithms: the energy-aware prefetching algorithm, and the energy-and-data aware prefetching algorithm.

1) Energy Model: In this paper, we use the energy model of LTE to formulate the energy consumption of prefetching. Currently, LTE is the fastest wireless network widely deployed. The power consumption of the LTE cellular interface can be generalized into three states: *promotion, data transmission*, and *tail*, as shown in Fig. 5. The power consumption of these three states are denoted as P_{pro}, P_{cell}, and P_{tail}, respectively. The promotion time T_{pro} and the tail time T_{tail} are fixed. The data transmission time depends on the size of the data and the network condition when the data is requested. To prefetch n ads, it takes time T_{pref}

$$T_{pref}(n) = \frac{n \times S_{ad}}{t_d}, \quad (1)$$

where S_{ad} is the ad size and t_d is the downlink throughput. Putting them together, the total energy of prefetching n ads through LTE can be formulated as

$$E_{pref}(n) = P_{pro} \times T_{pro} + P_{cell} \times T_{pref}(n) + P_{tail} \times T_{tail}. \quad (2)$$

To determine the power consumption of the LTE cellular interface, similar to previous work [27], we use the Monsoon power monitor as the power supplier for our Samsung Galaxy S6 phone (GS6) to measure the average instant power. The results are summarized in Table III, where the power consumption is measured when the screen is on.

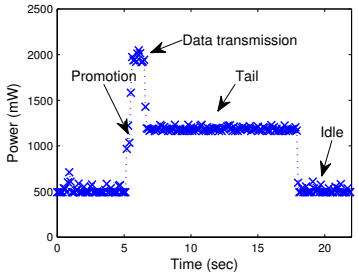


Fig. 5. Power level of using the LTE cellular interface to download an ad

TABLE III
POWER CONSUMPTION OF LTE CELLULAR INTERFACE

State	Power (mW)	Duration (s)
Idle (GS6)	498 ± 35.4	-
LTE Promotion	1286.3 ± 36.5	0.5 ± 0.1
LTE Data	1959.2 ± 42.1	-
LTE Tail	1192.4 ± 31.4	11.5 ± 0.9

2) Network Quality: The downlink throughput, t_d, is used as the indicator of the network quality. However, the information about the downlink throughput cannot be directly obtained from the smartphone. There are two methods to estimate the downlink throughput. The first method is based on the signal strength information such as the signal-to-noise ratio (SNR) or the signal-to-noise-plus-interference ratio (SNIR) [16]. However, this method is inaccurate and only used by simulators like ns3.

The second method is active probing which measures the downlink throughput by downloading some data [28], [29]. Fig. 6 shows the measured LTE downlink throughput with different amounts of data downloaded using a TCP connection. As shown in the figure, a large amount of data (i.e., more than 1 MB) is required to perform such measurement. This is because the time to download a small amount of data may depend on other factors rather than the downlink throughput, such as the initial congestion window, the slow-start mechanism of TCP, etc. Since the TCP slow-start only happens at the beginning of the data transmission, an enhanced method

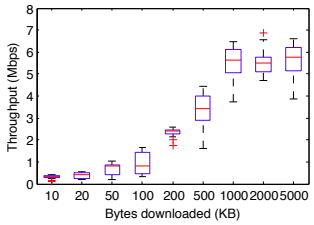


Fig. 6. Measuring LTE downlink throughput with different amounts of data. 1000 KB of data is required to accurately measure the downlink throughput of 5.5 Mbps.

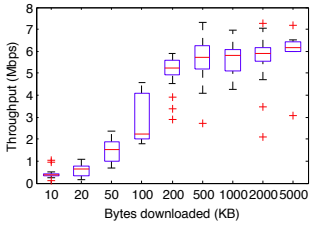


Fig. 7. Measuring LTE downlink throughput by using the second half of data. 200 KB of data is required to accurately measure the downlink throughput of 5.5 Mbps.

can divide the data in half and measure the throughput when downloading the second half of data. As shown in Fig. 7, by using the enhanced method, 200 KB data is enough to accurately measure the throughput.

3) Energy-aware Prefetching Algorithm: With multiple prefetching options provided by $\mathbb{R}$ and the current downlink throughput, the energy-aware algorithm aims to minimize the energy consumption of prefetching ads. To describe the algorithm, we introduce two variables: α_0 = 100%, $n_0 = 0$, which mean that more than 0 ads must be displayed for an app usage.

There are three cases when running the algorithm. 1) It is the first time to run the algorithm, and n_0 ad has been displayed. 2) n_j $(1 \leq j < M)$ ads have been displayed but insufficient (i.e., the prefetching option (α_j, n_j) has been chosen by the last run of the algorithm). 3) The displayed ads are equal to or more than n_M, which is the largest number of ads displayed in history.

The first two cases can be considered together. Given that n_p $(0 \leq p < M)$ ads have been displayed, the algorithm calculates the per-ad energy consumption ($E^p_{per\_ad}(i)$) for each prefetching option (α_i, n_i) such that $i > p$, and chooses the one with the minimum $E^p_{per\_ad}(i)$. To calculate $E^p_{per\_ad}(i)$, we first calculate the number of remaining ads, which is $(n_M - n_p)$. Then, we estimate the total energy consumption of prefetching the remaining ads, $E^p_{total}(i)$. Finally, we have

$$E^p_{per\_ad}(i) = \frac{E^p_{total}(i)}{n_M - n_p}. \tag{3}$$

To calculate $E^p_{total}(i)$, two parts of energy consumption need to be considered. The first part is the energy consumption of prefetching $(n_i - n_p)$ ads. This part of energy will be consumed immediately after the prefetching option (α_i, n_i) is chosen, and can be calculated by $E_{pref}(n_i - n_p)$ according to the current downlink throughput. The second part is the energy consumption of prefetching additional ads in the future, if and only if more than n_i ads are needed. This part of energy cannot be accurately calculated, because future prefetching behaviors are hard to predict.

Algorithm 1: Energy aware prefetching algorithm

Input : $\mathbb{R} = \{(\alpha_i, n_i) \mid i = 1, 2, ..., M\}$, Number of ads already displayed n_d, α_0 = 100%, $n_0 = 0$
Output : Number of ads to prefetch n_{pref}

```
n_pref ← 10;
E_min ← +∞;
if n_d ≥ n_M then
    return n_pref;
else
    find p s.t. n_p = n_d;
end
for i ← p + 1 to M do
    n ← n_i − n_p;
    pro ← α_i/α_p;
    E^p_total(i) ← E_pref(n) + pro × E_pref(n_M − n_i);
    E^p_per_ad(i) ← E^p_total(i)/(n_M − n_p);
    if E^p_per_ad(i) < E_min then
        n_pref ← n;
        E_min ← E^p_per_ad(i);
    end
end
return n_pref;
```

To simplify the calculation, we assume that if more than n_i ads are needed, one additional prefetching is used for all the remaining ads which is $(n_M - n_i)$. Since users usually stay still while using an app, the network quality should not change too much. Thus, the energy consumption can be calculated by $E_{pref}(n_M - n_i)$.

With the energy consumption for the current prefetching and future prefetching, we have

$$E^p_{total}(i) = E_{pref}(n_i - n_p) + pro \times E_{pref}(n_M - n_i), \tag{4}$$

where $pro = \alpha_i/\alpha_p$, which is the probability to display more than n_i ads after n_p ads already being displayed. If no ad has been displayed before (i.e., $p = 0$), $pro = \alpha_i$.

In case three, since the displayed ads are equal to or more than the largest number of ads displayed in history, we cannot decide how many ads to prefetch based on the historical app usages. Thus, we decide to prefetch a fixed number of ads. A complete description of the energy-aware prefetching algorithm is shown in Algorithm 1.

4) Energy-and-data Aware Prefetching Algorithm: Our energy-and-data aware prefetching algorithm also considers the effect of prefetching on the cellular data usage. In LTE, energy efficiency is preferred over data saving since big data plans are becoming more and more affordable. However, it is still necessary to provide a tradeoff between energy and data usage, because it is unreasonable to waste a large amount of data usage in exchange for a small improvement of energy efficiency.

The energy-and-data aware prefetching algorithm considers both per-ad data usage and per-ad energy consumption. Given that n_p $(0 \leq p < M)$ ads have been displayed, for each prefetching option (α_i, n_i) such that $i > p$, the per-ad data usage, $S^p_{per\_ad}(i)$, can be calculated as

$$S^p_{per\_ad}(i) = \frac{(n_i - n_p) \times S_{ad}}{N^p_e(i)}, \tag{5}$$

Algorithm 2: Energy-and-data aware prefetching algorithm

Input : $\mathbb{R} = \{(\alpha_i, n_i) \mid i = 1, 2, ..., M\}$, Number of ads already displayed n_d, α_0 = 100%, $n_0 = 0$, β
Output : Number of ads to prefetch n_{pref}

$n_{pref} \leftarrow 10$;
$C_{min} \leftarrow +\infty$;
if $n_d \geq n_M$ **then**
  return n_{pref};
else
  find p s.t. $n_p = n_d$;
end
for $i \leftarrow p+1$ **to** M **do**
  $n \leftarrow n_i - n_p$;
  $pro \leftarrow \alpha_i / \alpha_p$;
  $E^p_{total}(i) \leftarrow E_{pref}(n) + pro \times E_{pref}(n_M - n_i)$;
  $E^p_{per\_ad}(i) \leftarrow E^p_{total}(i)/(n_M - n_p)$;
  $N^p_e(i) \leftarrow [\sum_{j=p+1}^{i} (n_j - n_p) \times (\alpha_{j-1} - \alpha_j)]/(\alpha_p - \alpha_i)$;
  $S^p_{per\_ad}(i) \leftarrow [(n_i - n_p) \times S_{ad}]/N^p_e(i)$;
  $C^p_{per\_ad}(i) \leftarrow E^p_{per\_ad}(i) + \beta \times S^p_{per\_ad}(i)$;
  if $C^p_{per\_ad}(i) < C_{min}$ **then**
    $n_{pref} \leftarrow n$;
    $C_{min} \leftarrow C^p_{per\_ad}(i)$;
  end
end
return n_{pref};

where $N^p_e(i) = [\sum_{j=p+1}^{i} (n_j - n_p) \times (\alpha_{j-1} - \alpha_j)]/(\alpha_p - \alpha_i)$ is the expected number of ads that can be displayed among the prefetched $(n_i - n_p)$ ads. After obtaining $S^p_{per\_ad}(i)$, the per-ad cost, $C^p_{per\_ad}(i)$, can be calculated by adding $S^p_{per\_ad}(i)$ and $E^p_{per\_ad}(i)$ together with a parameter β to perform the tradeoff:

$$C^p_{per\_ad}(i) = E^p_{per\_ad}(i) + \beta \times S^p_{per\_ad}(i). \quad (6)$$

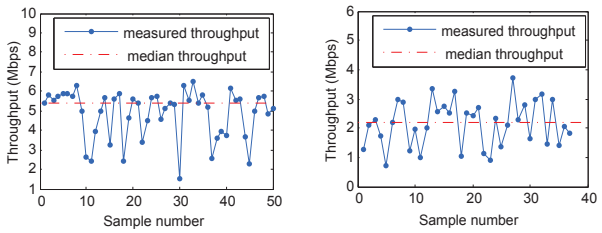
In the energy-and-data aware prefetching algorithm, the prefetching option that yields the smallest value of $C^p_{per\_ad}(i)$ is chosen. A complete description of the energy-and-data aware prefetching algorithm is shown in Algorithm 2.

V. Performance Evaluation

In this section, we first introduce the app usage traces and the network quality traces, and then use trace-driven simulations to evaluate the performance of our prefetching algorithms.

A. Evaluation Setup

In the evaluation, the ad refresh interval is set to two values: 30 seconds, which indicates an aggressive way to display ads, and 60 seconds, which is usually recommended by ad platforms like AdMob [22]. Three ad sizes are considered based on the discussion in Section III-A2: 50 KB, 100 KB, and 200 KB. We also compare four algorithms: "Non-prefetch" is the traditional way to periodically fetch ads, which is used as benchmark. "80th Percentile" indicates a prefetching algorithm that uses the 80th percentile value of the number of ads displayed in historical records to determine how many ads to prefetch, which is based on [4]. "Max" indicates a prefetching algorithm that prefetches the largest number of ads displayed in history. "Perfect" indicates a prefetching algorithm that can prefetch the exact number of ads that will be used by the app.



(a) Trace 1 (b) Trace 2

Fig. 8. Measurement-based throughput traces. Throughput is randomly collected when walking inside and outside the building.

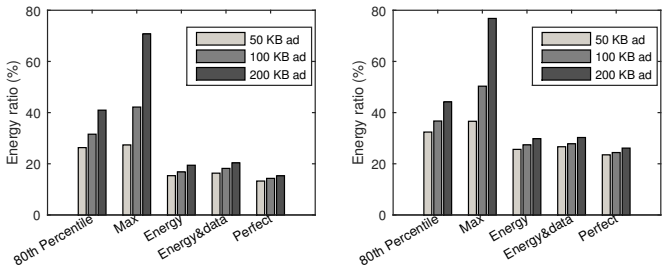
Note that the perfect algorithm does not exist, and it only provides a performance upper bound.

1) App Usage Trace: We use all 34 users and 20 apps for each of them from the LiveLab dataset to generate app usage traces. Each app usage trace contains the usage records of a certain app generated by a user. To train our prediction algorithm, each app usage trace is equally splitted into two parts. Half of the trace is used as the training data and the other part is used for evaluation.

2) Measuring Network Quality: We use our Samsung Galaxy S6 phone to collect throughput traces through a LTE network by downloading files from a mock server. Trace 1 in Fig. 8a is an outdoor trace with the downlink throughput averaged at 5.5 Mbps. Trace 2 in Fig. 8b is an indoor trace with downlink throughput averaged at 2.1 Mbps. The throughput of those two traces is much lower than that measured by speed test tools like Speedtest [30], which can reach 13 Mbps in the outdoor environment. The reasons are as follows: First, in our measurement, we use single HTTP connection over TCP, the same way of fetching ads, to download files. The TCP connection has been proven unable to fully utilize the LTE bandwidth, because its congestion control cannot quickly adapt to the network speed change. Second, speed test tools usually optimize connection parameters and use multiple threads to saturate the network connections. They also have a higher throughput than average.

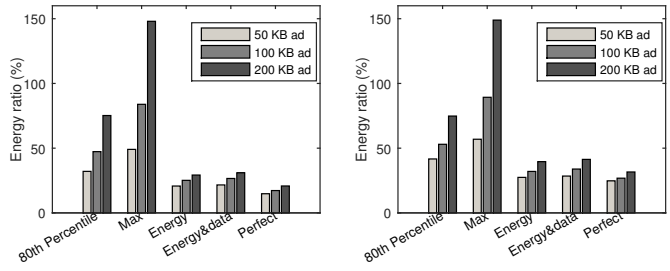
B. Evaluation Result (Measurement-based Network Quality)

Fig. 9 and Fig. 10 show the energy performance of different prefetching algorithms on two measurement-based throughput traces (i.e., Trace 1 and Trace 2). The energy consumption of the non-prefetch algorithm is used as benchmark to calculate the energy ratio of other algorithms. Based on the results, we have three observations. First, all prefetching algorithms save less energy when the ad size increases from 50 KB to 200 KB or when the network quality is poor (i.e., trace 2). This is because prefetching algorithms usually download more ads than needed. As the ad size increases or under poor network quality, more energy is wasted on downloading those unnecessary ads. Second, when the ad refresh interval is shorter (i.e., 30 seconds), more ads are displayed and there is more energy saving opportunity for prefetching. Third, our prefetching algorithms are comparable to the perfect algorithm on energy consumption, and outperform all other algorithms under various scenarios.



(a) 30 seconds ad refresh interval (b) 60 seconds ad refresh interval

Fig. 9. Energy ratio for measurement-based throughput (Trace 1)

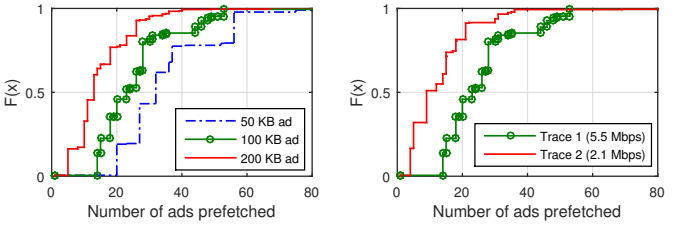


(a) 30 seconds ad refresh interval (b) 60 seconds ad refresh interval

Fig. 10. Energy ratio for measurement-based throughput (Trace 2)

To save energy, our algorithms can adjust the number of ads to prefetch according to the energy consumption of downloading an ad, which is influenced by the ad size and the network quality. Fig. 11a plots the CDF of the number of ads prefetched with different ad sizes according to our energy-aware prefetching algorithm. As can be seen, fewer ads are prefetched as the ad size increases from 50 KB to 200 KB. This is because more energy may be wasted on prefetching unneeded ads as the ad size becomes larger. Fig. 11b shows the number of ads prefetched under different network quality. Under good network quality (i.e., trace 1), prefetching unneeded ads consumes less energy, so our algorithms save energy by prefetching more ads to avoid other possible long tail problems. Since our prefetching algorithms can adaptively adjust the number of ads to prefetch, they exhibit steady performance on energy consumption under various scenarios. In contrast, the 80th percentile algorithm and the max algorithm cannot adjust the number of ads to prefetch according to the ad size and the network quality. So they consume much more energy under certain scenarios (e.g., 200 KB ad size, poor network quality).

The average data usage of different prefetching algorithms is shown in Fig. 12. The non-prefetch algorithm's data usage is used as benchmark to calculate the data ratio of other algorithms. As can be seen, the perfect algorithm has the least data usage because it only fetches the exact number of ads. Comparing with the energy-aware prefetching algorithm, whose data ratio is around 200%, the energy-and-data aware prefetching algorithm has a lower data usage and reduces its data ratio to around 170%. The data usage of the 80th percentile algorithm is acceptable, while the max algorithm uses an unaffordable amount of data which leads to a 1224% data ratio.



(a) Fewer ads are prefetched as the ad size increases from 50 KB to 200 KB.network quality is better. (100 KB ad) (Trace 1)

(b) More ads are prefetched when the ad(Trace 1)

Fig. 11. CDF plots of the number of ads prefetched according to the energy-aware prefetching algorithm (30 seconds ad refresh interval)

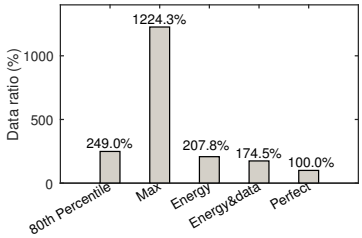


Fig. 12. Average data ratio

VI. Testbed Development and Evaluation

To understand how much energy can be saved by our prefetching algorithms in a real app, we have implemented the proposed algorithms and measured their performance in a book reader app on smartphones.

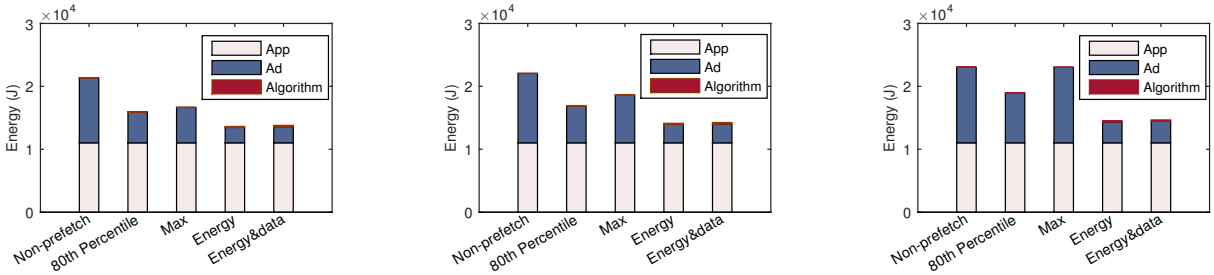
A. Testbed Development

We have developed three versions of a book reader app: *ad-enabled, ad-disabled*, and *no-ad*. The ad-enabled app embeds an ad client which can download ads from a mock server with different prefetching algorithms. The ad-disabled app embeds a tailored ad client which runs the prefetching algorithms without downloading ads. The no-ad app has no ad client. We distributed the app to seven students in our department, and collected app usage traces for one month. For each trace, half is used as the training data and the other half is used for evaluation.

We replay the traces using different versions of the app on a Samsung Galaxy S6 phone with 4G data plan, and use a Monsoon Power Monitor to measure the power consumption. The ad refresh interval is set to 60 seconds in our experiment. For ad-enabled and ad-disabled apps, we run multiple tests with different prefetching algorithms. Each test is repeated five times, and the average energy consumption is reported.

B. Experimental Results

The energy consumption of the ad-enabled app includes the energy consumed to prefetch ads (ad energy), the energy consumption of the prefetching algorithms, and the app energy (including CPU and display). The energy consumption of the ad-disabled app includes the energy of the prefetching algorithms and the app energy. The energy consumption of the no-ad app includes only the app energy.



(a) 50 KB ad (b) 100 KB ad (c) 200 KB ad

Fig. 13. Energy consumption in an app

By considering them together, we can break down the energy consumption of the ad-enabled app. As shown in Fig. 13, the energy consumption of our prefetching algorithms is insignificant compared to the ad energy. The ad energy of the non-prefetch algorithm (i.e., fetching ads periodically) accounts for almost half of the total energy consumed by the app. Compared to the non-prefetch algorithm, our prefetching algorithms can save 70% to 80% of the ad energy and 30% to 40% of the total energy with different ad sizes. Compared to the 80th percentile algorithm and the max algorithm, our prefetching algorithms can reduce the ad energy at least by half. When the ad size is 200 KB, the ad energy of our algorithms is one fourth of the ad energy of the max algorithm, and two fifth of the ad energy of the 80th percentile algorithm.

VII. Conclusion

In this paper, we proposed network quality aware prefetching algorithms to reduce the energy consumption of fetching in-app ads. First, in contrast to traditional data-mining based prediction algorithms, which only generate one option, we designed a prediction algorithm that generates a set of prefetching options with various probabilities. Second, with these prefetching options, we estimated the energy consumption of each option by considering the effect of network quality and chose the best one accordingly. Two prefetching algorithms were proposed: the energy-aware prefetching algorithm aims to minimize the energy consumption, and the energy-and-data aware prefetching algorithm also considers the data usage to achieve a tradeoff between energy and data usage. Evaluation results show that our prefetching algorithms can save 80% of energy compared to traditional ways of fetching ads periodically, and outperform existing prefetching algorithms under various network quality.

References

[1] "The history of app pricing and why most apps are free," http://blog.flurry.com/bid/99013/The-History-of-App-Pricing-And-Why-Most-Apps-Are-Free.

[2] "Download distribution of Android apps," http://www.appbrain.com/stats/android-app-downloads.

[3] S. Nath, "MAdScope: Characterizing Mobile In-App Targeted Ads," in *ACM MobiSys, 2015*.

[4] P. Mohan, S. Nath, and O. Riva, "Prefetching mobile ads: can advertising systems afford it?," in *ACM EuroSys, 2013*.

[5] B. Zhao, W. Hu, Q. Zheng, and G. Cao, "Energy-Aware Web Browsing on Smartphones," *IEEE Trans. Para. Dist. Syst.*, vol. 26, no. 3, pp. 761-774, 2015.

[6] Y. Geng, W. Hu, Y. Yang, W. Gao, and G. Cao, "Energy-Efficient Computation Offloading in Cellular Networks," in *IEEE ICNP, 2015*.

[7] W. Hu and G. Cao, "Energy Optimization Through Traffic Aggregation in Wireless Networks," in *IEEE INFOCOM, 2014*.

[8] W. Hu and G. Cao, "Energy-Aware Video Streaming on Smartphones," in *IEEE INFOCOM, 2015*.

[9] F. Qian, Z. Wang, Y. Gao, J. Huang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Periodic transfers in mobile applications: network-wide origin, impact, and optimization," in *ACM WWW, 2012*.

[10] "Configuration of fast dormancy. rel. 8," http://www.3gpp.org/.

[11] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft, "Breaking for commercials: characterizing mobile advertising," in *ACM IMC, 2012*.

[12] 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 *ACM EuroSys, 2012*.

[13] A. Parate, M. Böhmer, D. Chu, D. Ganesan, and B. M. Marlin, "Practical prediction and prefetch for faster access to applications on mobile phones," in *ACM UbiComp, 2013*.

[14] B. D. Higgins, J. Flinn, T. J. Giuli, B. Noble, C. Peplin, and D. Watson, "Informed mobile prefetching," in *ACM MobiSys, 2012*.

[15] Y. Wang, X. Liu, D. Chu, and Y. Liu, "EarlyBird: Mobile Prefetching of Social Network Feeds via Content Preference Mining and Usage Pattern Analysis," in *ACM MobiHoc, 2015*.

[16] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan, "Bartendr: a practical approach to energy-aware cellular data scheduling," in *ACM MobiCom, 2010*.

[17] A. J. Khan, K. Jayarajah, D. Han, A. Misra, R. Balan, and S. Seshan, "CAMEO: a middleware for mobile advertisement delivery," in *ACM MobiSys, 2013*.

[18] Y. Chen, P. Berkhin, B. Anderson, and N. Devanur, "Real-time bidding algorithms for performance-based display ad allocation," in *ACM SIGKDD, 2011*.

[19] "Three trends in mobile advertising," http://mobiledevmemo.com/three-trends-in-mobile-advertising/.

[20] "Millennial media ad specs," http://www.millennialmedia.com/ad-specs.

[21] "Yahoo ad specs," https://adspecs.yahoo.com/adformats/mobile.

[22] "AdMob ad specs," https://support.google.com/admob.

[23] "iAd ad specs," https://developer.apple.com/news-publisher/iad/Creative-Specifications.pdf.

[24] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum, "LiveLab: measuring wireless networks and smartphone users in the field," in *ACM HotMetrics, 2010*.

[25] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a data set via the gap statistic," *Journal of the Royal Statistical Society B*, vol. 63, no. 2, pp. 411-423, 2001.

[26] R.E. Neapolitan, *Learning Bayesian Networks*, Prentice Hall, 2004.

[27] Y. Yang, Y. Geng, L. Qiu, W. Hu, and G. Cao, "Context-aware task offloading for wearable devices," in *IEEE ICCCN, 2017*.

[28] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck, "An in-depth study of LTE: effect of network protocol and application behavior on performance," in *ACM SIGCOMM, 2013*.

[29] W. Li, R.K.P. Mok, D. Wu, and R.K.C. Chang, "On the accuracy of smartphone-based mobile network measurement," in *IEEE INFOCOM, 2015*.

[30] "Ookla Speedtest," http://www.speedtest.net/mobile/.