Capitalizing on the Promise of Ad Prefetching in Real-World Mobile Systems

Yimeng Wang, Yongbo Li, Tian Lan
George Washington University, Washington, DC
{wangyimeng, lib, tlan}@gwu.edu

***Abstract*—In cellular networks, tail states are designed for a tradeoff between energy efficiency and latency. However, the energy consumed during tail states becomes a huge energy drainer itself. Traditional energy saving techniques by content prefetching cannot be directly applied to mobile ads, due to the deadline requirements of ads, randomness in user behaviors, different usage patterns of mobile apps and system services. In this paper, considering several significant runtime factors, we make a novel use of Markov Decision Process to model the energy minimization problem for ad prefetching (EMAP), and propose an algorithmic solution to the EMAP problem. Further, we implement the first mobile ad prefetching system that is fully compatible with contemporary ad libraries and mobile apps. By replaying real-world user traces on Android devices, we show our proposed solution consistently outperforms existing On-Demand policy on Android by up to 59% in saving ad-related energy, while a simple Fill-Up-Buffer policy can be even 3 times worse than the default On-Demand policy. Such findings provide critical insights regarding the promise of saving energy by ad prefetching in real-world mobile systems.**

I. Introduction

In contemporary cellular networks, tail states are designed to guarantee the responsiveness of network connections. In particular, after active data transmission terminates, the network interface normally stays in an intermediate power state for a certain period (e.g., 11.6 seconds [16] when using LTE networks) before returning to an idle state. While these tail states introduce a tradeoff between energy and responsiveness, they have become a big energy drainer for mobile systems.

Recent research findings [16] show that on average tail states consume 48.2% and 48.1% of transmission energy for LTE and 3G networks. The energy inefficiency problem further stands out when the mobile traffic is periodic and the data transmission has short durations, resulting in a significant amount of time in intermediate power states. For instance, mobile ads often refresh every 12 to 120 seconds [19]. This leads to a traffic pattern that periodically fetches only a small amount of data, which means short active transmission time, yet keeps pushing the network interface to high power states, generating a large amount of tail energy. To mitigate such problem, content prefetching (e.g. email, webpage) has been proposed as a promising energy saving technique in existing work [29], [14], [22], as well as advocated by mobile operating system developer [4] and Internet service provider [13].

However, traditional content prefetching techniques are not directly applicable to mobile ad prefetching. Existing work even provides contradicting conclusions on the feasibility of ad prefetching in real-world mobile networks (e.g., prior work in [20] shows communication energy consumed by ads can be reduced by more than 50% by ad prefetching, which amounts to 17% of total energy. In contrast, results in [8] claim 3G tail energy only constitutes on average 3.2% of the total energy and this is the hard-to-achieve upper bound for ads energy saving techniques). The main challenges are two-folds: First, mobile ads are time-sensitive, usually with deadlines imposed by advertisers. A naive approach that prefetches and caches a large number of ads may not only violate the service-level agreement (SLA) between ad networks and advertisers [19], but also cause excessive energy waste due to over-aggressive ad fetching. Second, a significant amount of randomness exists in real-world mobile systems, such as in user behavior, network uncertainty, and other runtime system dynamics. To be specific, users and apps may exhibit very different patterns of ad usage, e.g., in terms of ad refreshing rates, which has a direct impact on the required aggressiveness of ad prefetching. On the other hand, non-ad traffic generated by background services or apps unavoidably changes the status of network interfaces, which in turn varies the energy efficiency of prefetching. Blindly prefetching a fixed number of ads [19] in a pre-determined fashion fails to adapt to system randomness. Without a systematic optimization framework that is backed up by a full-scale prototype, it remains an open problem whether it is indeed feasible to capitalize on the promise of ad prefetching techniques for saving mobile energy.

Toward this end, we propose an optimization framework that comprehensively tackles the unique challenges rising from ad prefetching in the real-world mobile system. We make use of Markov Decision Process (MDP) [15] to systematically model ad expiration, runtime dynamics, and diversity in mobile user/app behavior as system state transitions. In particular, we consider system states consisted of a three-tuple - (i) status of network interface, (ii) the active app in use, and (iii) the number of residue ads currently available in the system. Characterizing the transitions between these system states, we formulate an MDP with an energy-aware reward function to minimize the expected energy consumption over all states. The proposed optimization problem is then solved by an efficient algorithm. To the best of our knowledge, this is the first systematic framework to optimize ad prefetching over both user and system dynamics jointly. The proposed solution enables customized ad prefetching strategy that adaptively adjusts to the system and mobile user behaviors to minimize

energy drain due to ad modules in mobile networks.

We fully prototype our framework on Android with the algorithmic solution and a proxy for ad prefetching, which are the key modules of our implementation. Our system listens to the runtime state changes triggered by three types of events: (i) an ad display, (ii) switches between working states of network interfaces, and (iii) the changes of foreground app. The state changes are monitored, and a database storing the optimal ad prefetching decisions (computed using our proposed MDP model and optimization algorithm) is queried on the fly. All prefetched ads are cached in our ad proxy and displayed to mobile apps in a seamless manner, transparent to contemporary ad libraries. The fully implemented system solution is evaluated using 8 real-world user traces [2]. A significant amount of ad-related energy saving is validated, when compared with the default 'On-Demand' policy of contemporary ad libraries [1] and a heuristic 'Fill-Up-Buffer' policy.

The main contributions of our work, spanning theoretical modeling and system implementations, are summarized below:

1) We propose a novel approach to model and optimize mobile ad prefetching in real-world networks by a Markov Decision Process, considering several important and realistic factors of user and runtime dynamics. We further propose an algorithmic solution to the optimization problem, which is shown to be effective and consistent.

2) The MDP-based ad prefetching system, including an optimization engine and an ad proxy, is prototyped on Android. To the best of our knowledge, this is the first fully implemented ad optimization system that is compatible with contemporary ad libraries and mobile apps in a transparent manner.

3) Using real-world apps and 8 user traces from LiveLab [2], we evaluate the proposed solution and system prototype. We show that the proposed MDP-based approach consistently outperforms existing On-Demand policy by up to 59% in ad-related energy saving, and achieves 3 times energy efficiency compared to a Fill-Up-Buffer policy. Such findings provide critical insights regarding the feasibility and optimization of ad prefetching in real-world mobile systems.

It is worth mentioning that our proposed MDP-based optimization and the system implementation is not limited to mobile ads traffic. Rather, it is easily extensible to other types of contents, which makes it a potential technique to explore more energy saving opportunities further.

II. Problem Statement

In this section, we introduce background on cellular network tail energy and ad prefetching. Then, we use an illustrative example to motivate the necessity of optimized mobile ad prefetching decisions for energy saving.

A. Tail Energy and Prefetching

To minimize latency, after data transmission is finished, the transition to low power states and idle states has been both postponed for certain durations (*tail time*) in contemporary 3G and 4G cellular networks [16]. The energy consumed during the tail time (*tail energy*) can account for a large portion of the overall network energy, especially for periodic and short data transmissions. Mobile ad libraries typically refresh displayed ads every 12 - 120 seconds (e.g., 60 seconds default period is used by the popular ad library AdMob [1]), however, the actual time consumed for ad data transmission is very short.

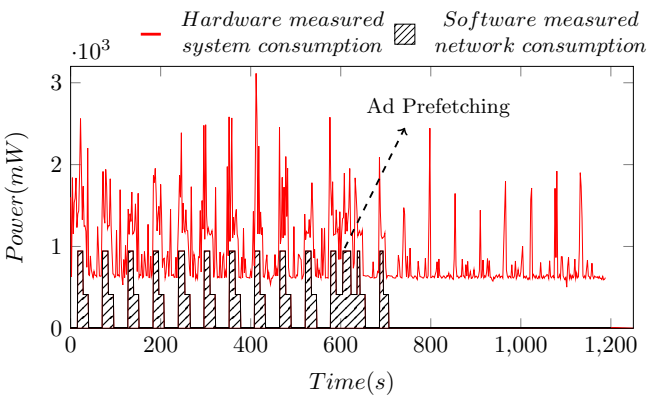


Fig. 1. System power (top) measured by Monsoon Power Monitor along with network interface power (bottom) measured by software-based Energy Meter.

Content prefetching is advocated by both mobile system developers [4] and Internet service providers [13] for energy efficiency. We use an app (with ad module by AdMob library) to illustrate the effects of prefetching on energy consumption. The app is an offline tool and utilizes Internet service only for ad data transmission every 1 minute. We use our proposed ad prefetching system (with details in Section V) to prefetch 10 ads every 10 minutes. We connect the phone to external Monsoon Power Monitor [3] to measure the system power consumption and utilize a software-based power meter [31] to measure the network energy consumption. The power consumption traces are depicted in Figure 1. As seen in the figure, in the first 10 minutes, ads are fetched on-demand. The hardware measured energy shows that each time an ad is requested, relatively consistent amount of energy is consumed, and the software-based measurement shows a tail is triggered. After 10 minutes, a relatively large amount of energy is consumed in a short time (marked by dashed arrow in the figure); and this is when prefetching happens. Afterward, when an ad is requested again, the whole system power consumption is much smaller than that in the first 10 minutes, and no network energy tail is generated.

The above simple example shows the effectiveness of ad prefetching in saving energy. However, in real-world scenarios, traditional content prefetching techniques [29], [14], [22] cannot be directly applied to mobile ad prefetching since ads are always imposed with the deadline requirements from advertisers, and prefetching policies that are unaware of the ad deadlines will unavoidably result in SLA violations. In the following, we use a concrete example to motivate the necessity for an optimized prefetching policy.

B. Motivational Example

With the ad deadline taken into account, an intuitive goal is to prefetch the 'right' amount of ads and guarantee they are displayed before the deadlines. Consider the following three possible ad fetching policies:

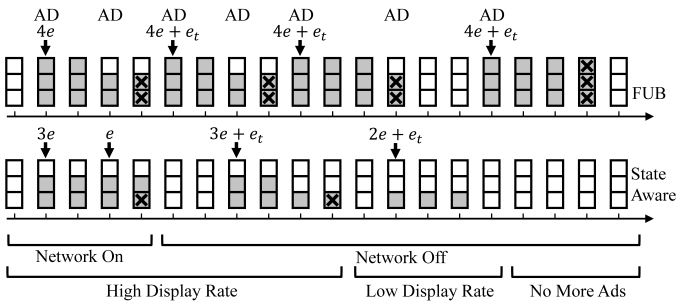


Fig. 2. Illustration of the prefetched ads and consumed energy in a mobile system with different states for the motivational example

***On-Demand**:* This is the default policy of contemporary ad libraries. When the running app requests an ad, the ad library handles the request in an on-demand manner with the ad servers, which generates network traffic for each ad request.
***Fill-Up-Buffer**:* This policy heuristically pre-set a buffer size, and prefetches a batch of ads every time the buffer becomes empty. Prior work [19] uses a similar policy which pre-sets the prefetching period and aggressiveness by heuristics.
***State-Aware**:* This policy considers expiration rate, ad display rate, and network status as primary factors and adjust the prefetching decisions accordingly.

Consider the following example, when a user uses two apps: one has a slightly higher ad display rate of 1 every 2 iterations while the other one has lower ad display rate of 1 every 3 iterations. To simplify the example for illustration purpose, the network state is either on or off. When the network interface is on due to background traffic, fetching ads from the server will generate no additional energy tail; while the background traffic is off, an energy tail (e_t) will be added to the fetching energy. For this example, we assume each ad expires after three iterations from when it is prefetched.

The illustration for the numbers of prefetched ads and energy consumption for the Fill-Up-Buffer and State-Aware policies is depicted in Figure 2. In the figure, iterations marked by "AD" indicates that one ad is requested and displayed by the currently active app. The three blocks denote the buffers in the system used for storing ads, and a gray block indicates the block is occupied by a prefetched ad. A block with an 'X' mark denotes that at the end of the iteration, the ad in the block will expire. It's clear that On-Demand (OD) policy will consume $7e + 5e_t$ of overall energy and the illustration is left out from the figure.

The energy consumed to fetch ads is marked along the arrows. With the buffer size equals to three, the Fill-Up-Buffer (FUB) policy fetches four ads at the first iteration, with one for immediate display. Since the network is on, the energy consumed here is $4e$ where e is used to denote the average energy consumed for transmitting the data for a single ad. In the 4-th iteration, two ads will expire, so another four ads are prefetched and the consumed energy equals to $4e + e_t$ since the network is off. FUB policy repeats this behavior and the total energy consumed is $16e + 3e_t$.

In contrast, the State-Aware policy dynamically adapts its prefetching actions to the changes of system states. When the network is on, it becomes more aggressive and maintains two available ads when the display rate is high and one available ad for low display rate. With the network off, due to larger marginal cost by ad prefetching, it becomes conservative and prefetches two ads only when all ads are used up. In summary, State-Aware policy only fetches 9 ads with much less number of ads expired. The overall energy consumption is $9e + 2e_t$.

C. Design Goal

The simplified example above shows that in simple heuristic solutions, unawareness of the system states and the ad expiration will not only result in expired ads violating SLA, it also consumes more energy (can be even worse than the default On-Demand manner).

To minimize the energy consumed by ads and avoid possible ad expirations, our goal in this paper is to make optimized prefetching decisions, with different system states taken into considerations.

III. System Model

We consider a mobile system with a set $\mathbf{A}$ of installed apps that display ads to the users (also termed *ad-embedded apps* in this paper). The average ad display rate for app $a \in \mathbf{A}$ is represented by λ_a^d, which may vary for different apps. For example, app developers may incorporate a banner for showing ads and adopt the default refreshing rates for different ad libraries. Ads may also be refreshed only under certain triggers, e.g. a game app may show an ad after the user enters the next trial, and a news app shows an ad each time a new page is opened. The display rate of currently active app affects the number of ads needed. With the varied refreshing rates λ_i^d, we model the ad arrivals as a separate Poisson process for the app currently active in the foreground.

We define the network state as n, and $\mathbf{N}$ is the enumeration of all the possible states of n. Without loss of generality, $\mathbf{N}$ can be defined as {*High Power*, *Low Power*, *Standby*} and

$$n = \begin{cases} 1, & High\ Power\ state, \\ 2, & Low\ Power\ state, \\ 3, & Standby\ state. \end{cases} \quad (1)$$

In specific, in accordance with the state machines of 3G and 4G networks [16], High Power state corresponds to *Short DRX* and *Long DRX* in *RRC_CONNECTED* for LTE network, and *DCH* for 3G network. Low Power state is an intermediate state between Standby (*IDLE*) and High Power state. Besides, the delays for state transitions are set by different types of networks and service providers. For a given network type and service provider, we can further define the tail energy consumed during the timeout from High Power state to Low Power state, and from Low Power state to Standby state as e_1 and e_2, respectively. We use $e(x)$ to denote the energy consumed by fetching the data for x number of ads. Network status n and x jointly determine the overall marginal cost $f(x, n)$ generated to fetch a batch of ads:

$$f(x,n) = \begin{cases} e(x) & n = 1, \\ e(x) + e_2 & n = 2, \\ e(x) + e_2 + e_1 & n = 3 \end{cases} \quad (2)$$

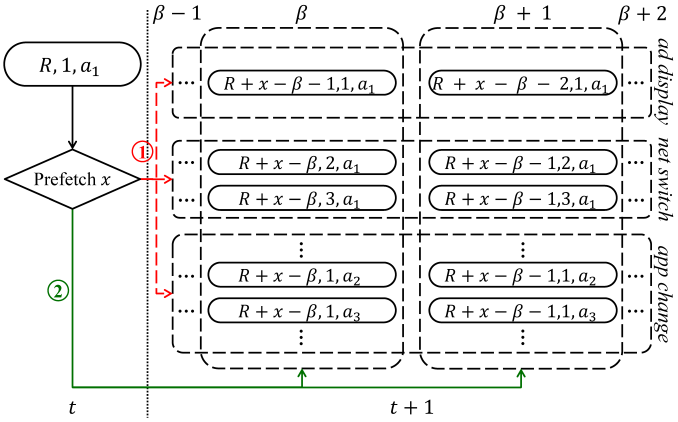


Fig. 3. State (denoted by rounded rectangles) transition when any of the three trigger events {ad display, net switch, app change} occurs. The next state is jointly determined by the event (marked by ①), expired ads β (marked by ②), and action x.

In consideration of ad deadlines, the number of available residue ads in the system has a direct effect on future prefetching decisions, and we denote it as r.

With the notations introduced above, we can uniquely represent the ***system state*** by a three-tuple variable s:

$$s = (r,\ n,\ a), \tag{3}$$

and the set of all possible states is $\mathbf{S}$. For any current state denoted as $s_i \in \mathbf{S}$, we consider a set $\mathbf{V}$ of ***trigger events***:

$$\mathbf{V} = \{d,\ m,\ u\}, \tag{4}$$

in which, the d, m, and u correspond to (i) an ad is displayed, (ii) the network status changes, and (iii) the currently active app is switched. The occurrence of any of the three trigger events will inevitably affect the state.

The goal of this paper is to find an optimal number of ads x to prefetch for each state $s \in \mathbf{S}$, so that the expected energy consumed by ads over all the possible states is minimized. The action x is taken when any state comes to an end due to the trigger events $\mathbf{V}$.

IV. Our Approach

Considering the possible states $\mathbf{S}$ in a mobile system, we first formulate the Energy Minimization for Ad Prefetching (EMAP) problem based on a novel use of Markov Decision Process (MDP). Then, we derive the optimal solution and further propose an algorithmic solution to the EMAP problem.

A. Modeling the State Transition

As discussed in Section III, any trigger event in $\mathbf{V}$ will affect the current state denoted as $s_i \in \mathbf{S}$. Our goal is to make an optimal action x when any state comes to an end due to the above system events. Though ad expiration is another event that may affect the state, consideration of the effects of expired ads is not necessary until the occurrence of new trigger events. The diagram of state transition and action taken is depicted in Figure 3, in which each rounded rectangle denotes a unique state, β denotes the number of ads expired between two states, and the dashed arrows correspond to the three trigger events we considered. When any of the events occurs, an action x is immediately taken. The next state is jointly determined by the event (marked by ①), expired ads β (marked by ②), and action x. For example, with action x when network status switches from 1 to 2, and β ads expired, the state will transits from current $(R, 1, a_1)$ to $(R + x - \beta, 2, a_1)$.

B. Constructing Transition Probabilities

As introduced in Section III, the ad display events for an active app a can be represented by a Poisson process with rate denoted by λ_d which varies for different apps [1]. Besides, as studied in prior work [11] on mobile usage, user tends to have different session (the time between launching and closing an app) lengths for different apps, we introduce λ_u as the rate of app changes when the active app is a, and similarly λ_m as the rate of network status switches. Without loss of generality, we assume the three trigger events do not happen simultaneously. It is easy to consider simultaneous events as a serial of consecutive events. Now we can denote the length of individual states as an exponentially distributed variable with mean equals to $1/\lambda$ where $\lambda = \lambda_d + \lambda_u + \lambda_m$. λ is the overall arrival rate for the trigger events when the active app is a. As mentioned above, the ad expiration can be considered as a separate event along with the state changes, and we define it as a Poisson process with expiration rate λ_b.

Ad Expiration: Considering the trigger events and ad expiration as two separate processes with rates λ and λ_b, one single ad's expiration can be modeled as a Bernoulli distribution, with the probability equal to that of ad expiration arriving earlier than trigger events, i.e. $\frac{\lambda_b}{\lambda_b+\lambda}$. Further, since different ads' expirations are independent and identically distributed, the probability that β out of r ads expire before next state is

$$p_e(\beta, r) = \begin{cases} \binom{r}{\beta}\left(\frac{\lambda_b}{\lambda_b+\lambda}\right)^{\beta}\left(\frac{\lambda}{\lambda_b+\lambda}\right)^{r-\beta} & 0 \le \beta \le r \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Trigger Events: Given λ as the arrival rate of all trigger events, we analyze the probabilities of each type of trigger events in the following.

I: Network Switch: Under the condition that trigger events occur when app a is active, the probability of network status switches is

$$p_m = \frac{\lambda_m}{\lambda}. \tag{6}$$

Under the condition that network switch event happens when current network status is n_i, the probability that the next network status is n_j can be denoted as $p(n_i \to n_j | m)$. Recall that we assume a state change is triggered by single event and multiple trigger events will generate consecutive state changes, hence, when net switches, the app a stays the same, and no ad is displayed. The change in number of residue ads $r_i \to r_j$ is due to expired ads $\beta_{i\to j}$ and prefetched ads. When action x is taken, $\beta_{i\to j} = r_i + x - r_j$, and the probability that state

[1]For simplicity, we suppress the subscript a representing the currently active app in Section IV without causing any confusion.

$s_i = (r_i, n_i, a)$ transits to $s_j = (r_j, n_j, a)$ when network switch happens is obtained:

$$p(i \to j|x, m) = p_e(\beta_{i\to j}, r_i + x) \times p(n_i \to n_j|m) \quad (7)$$

II: App Change: When user changes the app currently active in the foreground, the system transits to the next state. Similar to network switch event, the probability of app changes is:

$$p_u = \frac{\lambda_u}{\lambda}, \quad (8)$$

and the probability that state s_i transits to s_j is:

$$p(i \to j|x, u) = p_e(\beta_{i\to j}, r_i + x) \times p(a_i \to a_j|u) \quad (9)$$

III: Ad Display: The probability of ad display represented by λ_d and λ is

$$p_d = \frac{\lambda_d}{\lambda}. \quad (10)$$

After x ads are prefetched, one ad is displayed before transiting to state s_j, and we get the following transition probability:

$$p(i \to j|x, d) = p_e(\beta_{i\to j}, r_i + x) \quad (11)$$

However, for the above equation, it is necessary to distinguish two cases exist for the number of ads expired $\beta_{i\to j}$: (i) $\beta_{i\to j} < r_i + x$, one ad from the $r_i + x$ ads is displayed, so the number of expired ads can be calculated as $\beta_{i\to j} = r_i + x - r_j - 1$; (ii) $\beta_{i\to j} = r_i + x$, meaning all the previous r_i residue ads and the prefetched x ads expired before being displayed and the one ad displayed need to be fetched again.

C. Calculating the Cost for Actions

The taken actions x will affect the cost (termed reward in MDP [7]). In the following, we quantify the cost by action x for each of the state transitions discussed above.

As introduced in Section III, the cost of fetching x ads depends on the network state n. Equation (2) can be used as the cost function when the state transition is triggered by Net Switch $c(x|m)$ or App Change $c(x|u)$.

$$c(x|m) = c(x|u) = f(x, n) \quad (12)$$

However, special considerations are needed when ad is displayed. We again distinguish the two cases regarding the values of $\beta_{i\to j}$ and $r_i + x$: *Case 1:* $\beta_{i\to j} < r_i + x$ and *Case 2:* $\beta_{i\to j} = r_i + x$. We further use p_1 and p_2 to denote the probabilities of the two cases, i.e. p_1 and p_2 respectively denotes the probability that the displayed ad is from the $r_i + x$ ads and the displayed ad is fetched on-demand due to the expiration of all $r_i + x$ ads. p_1 and p_2 can be obtained by:

$$p_1 = \frac{\sum_{r_j} p(r_i + x - r_j - 1, r_i + x)}{\sum_{r_j} p(r_i + x - r_j - 1, r_i + x) + p(r_i + x, r_i + x)}$$
$$p_2 = \frac{p(r_i + x, r_i + x)}{\sum_{r_j} p(r_i + x - r_j - 1, r_i + x) + p(r_i + x, r_i + x)} \quad (13)$$

Considering the effects on costs from network state n, now we can summarize the cost when state transition is triggered by ad display as:

$$c(x|d) = \begin{cases} e(p_1 x + p_2(x+1)), & n = 1 \\ e(p_1 x + p_2(x+1)) + p_2 e_2], & n = 2 \\ e(p_1 x + p_2(x+1)) + p_2(e_2 + e_1), & n = 3 \end{cases} \quad (14)$$

D. Formulating the EMAP Problem

With the state transition probabilities and energy cost functions constructed, now we formally formulate the Energy Minimization for Ad Prefetching (EMAP) Problem.

We consider to minimize the expected accumulated cost over time with a discount factor γ for future time. γ discounted method [7] is generally used in MDP formulation to discriminate the relative importance between future rewards and current rewards. Denote the state at t and $t+1$ by s_t and s_{t+1}, and $c_{s_t, s_{t+1} \in \mathbf{S}}(s_t \to s_{t+1}|x)$ represents the cost when state transits from s_t to s_{t+1} under the condition that action x is taken. The objective is to find the optimal x_s for all $s \in \mathbf{S}$, such that the expected cost overall all the considered current and future time slots are minimized:

$$\begin{aligned}
&\text{Problem EMAP :} \\
&\min \quad \sum_{t=1}^{\infty} \gamma^t E[c(s_t \to s_{t+1}|x_s)] && (15) \\
&\text{s.t.} \quad E[c(s_t \to s_{t+1}|x)] \\
&\qquad = \sum_{s_t, s_{t+1} \in \mathbf{S}} p(s_t \to s_{t+1}|x) c(s_t \to s_{t+1}|x) && (16) \\
&\qquad p(s_t \to s_{t+1}|x) = \sum_{v \in \mathbf{V}} p_v \cdot p(s_t \to s_{t+1}|x, v) && (17) \\
&\qquad c(s_t \to s_{t+1}|x) = \sum_{v \in \mathbf{V}} p_v \cdot c(x|v) && (18) \\
&\qquad Equations\ (5) - (14) && (19) \\
&\text{var.} \quad \{x_s,\ \forall s \in \mathbf{S}\}. && (20)
\end{aligned}$$

Note that the equation (18) is simplified, making use of the cost models we derived in Section IV-C. $c(x|v)$ for each $v \in \mathbf{V}$ also depends on the state transition between s_t and s_{t+1}, which correspond to i and j in equation (13).

E. Algorithmic Solution to EMAP Problem

Now we present an algorithmic solution to the formulated EMAP Problem based on Value Iteration [7]. Assume c_j^* is the minimal cost value can be achieved at a future state s_j, then the cost $c(s_i, x)$ during state s_i is:

$$c(s_i, x) = c_{imd}(s_i, x) + \gamma \sum_{s_j \in S} p(s_i \to s_j|x) c_j^*, \quad (21)$$

with the left and right parts being the immediate and future costs, respectively, hence the optimal value and decision for state s_i are:

$$c_i^* = \sup_x c(s_i, x); \quad x_i^* = \arg_x [c(s_i, x) = c_i^*] \quad (22)$$

Besides, inspired by the diminishing effect over future cost imposed by the discount factor γ, we further adopt too heuristics methods to reduce the complexity of the algorithm. As seen from optimization objective (15), the expected cost is averaged over all future times over an infinite horizon. In reality, due to γ, the future costs may diminish quickly as the number of future iterations increases. In our algorithm, we keep tracking the future discounted cost, and once the cost reduces to a certain ratio of the immediate cost, the future iterations of calculations are canceled. Further, we set a limit

$\mathbf{M} \leftarrow$ HashMap from states s_i to minimal cost and decision (c_i^*, x_i^*)
$\mu \leftarrow$ Stop threshold for future iterations
$\mathbf{I} \leftarrow$ Max number of iterations allowed
$\mathbf{X} \leftarrow$ Max number of ads can be prefetched
for all states $s_i \in \mathbf{S}$
  if $s_i \notin \mathbf{M}$
    (c_i^*, x_i^*) = ValueIteration(s_i, 0, $+\infty$), add [s_i, (c_i^*, x_i^*)] to $\mathbf{M}$
  end if
end for

function ValueIteration(s_i, $iter$, $first\_cost$)
  for prefetch decision $x \in [0, X]$
    Get immediate cost $c_{imd}(x)$ (Equation (6), (8), (10), (12), (14))
    if $iter$ == 0 $first\_cost = c_{imd}(x)$ **end if**
    $iter++$
    if $iter > \mathbf{I}$
      Future cost $c_{ftr} = 0$
    else
      Get transition probabilities $p(i \to j|x)$ (Equation (5) - (12))
      if $s_j \notin \mathbf{S}$
        (c_j^*, x_j^*) = ValueIteration(s_j, $iter$, $first\_cost$)
        add [s_j, (c_j^*, x_j^*)] to $\mathbf{M}$
      end if
      Future cost $c_{ftr} = \sum_{s_j \in \mathbf{S}} \{p(i \to j|x) \cdot c_j^*\}$
    endif
    if $c_{ftr} < \mu \cdot first\_cost$ **break**
    Get expected total cost $c(s_i, x) = c_{imd}(x) + \gamma c_{ftr}$
  end for
  Get optimized cost $c_i^* = \min_x c(s_i, x)$
  return (c_i^*, x_i^*)
end function

Fig. 4. Pseudo code of our algorithmic solution to Problem EMAP

on the max number of iterations within considerations. In these ways, the computation complexity is largely reduced without sacrificing the performance of the solution. The algorithmic solution to Problem EMAP is summarized in Figure 4.

V. Design of EMAP System

This section provides our system design and implementation details. The prototype of the ad prefetching system is implemented on Android, with the key modules and work flow depicted in Figure 5. We dedicate each following subsection to each of the key modules.

A. Ad Proxy

Ad prefetching and ad requests are handled by a Proxy-based module in our design. We utilize iptables to redirect ad-related HTTP and HTTPS traffic to our Proxy. The HTTPS traffics are handled with a pre-installed certificate on the system. To reduce the overhead on the network performance, we only filter out the ad traffics based on a whitelist of IP addresses. The list is constructed using several popular ad libraries including the popular ad library AdMob [1] on Android platform. When prefetched ads are available in the system, the ad requests from apps are handled in an offline manner without changing the status of network interfaces.

The Proxy module is also responsible for recording relevant information needed by other modules. For example, to be able to calculate the energy consumed by ads, the Proxy keep tracks of the timestamps of ad traffic. Since Proxy directly handles the ad traffics, it is also responsible for sharing the information

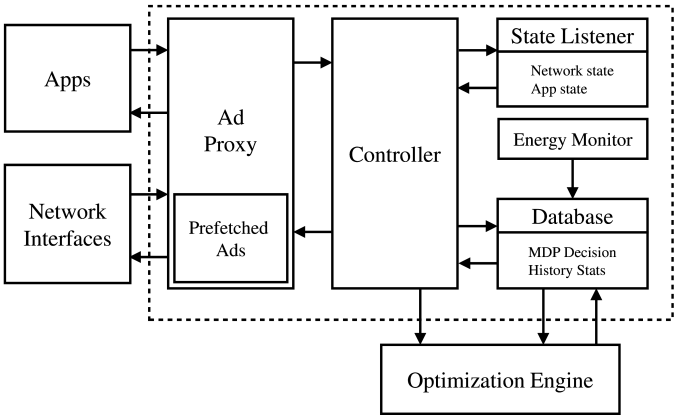


Fig. 5. Overview of EMAP System

with our State Listener module when an ad is displayed, i.e. the occurrence of the trigger event d we introduced in Section III.

B. State Listener

State Listener is a unified module we implement to listen to each of the trigger events in $\mathbf{V}$. The workload of updating state and notifying the centralized Controller is implemented as callback functions. As mentioned before, when an ad is requested, the Proxy will trigger the callback function. Similarly, when the network status n changes, another module Energy Monitor (which will be discussed later) is responsible for notifying the changes. Detection of the changes of app state a is implemented as a separate background service to periodically query the package name of the foreground app, and the service works seamlessly with State Listener.

C. Controller

Controller is the centralized module that coordinates the other modules. First, once it is notified by the State Listener of any state changes, it will work with Optimization Engine to make an optimal decision using the algorithm we discussed in the last Section and send the decision to Proxy if any ad prefetching task is needed. The Optimization Engine also needs the information shared by the Controller such as the current system states.

D. Optimization Engine

The algorithmic solution to Problem EMAP, with pseudo-code listed in Figure 4 is implemented in Optimization Engine. Optimization Engine is responsible for performing all the required calculations needed in the algorithm, such as calculating the transition probabilities, quantifying the costs for different state transitions and decisions x. It is worth mentioning that the calculations and optimization work is not needed every time the state changes. Rather, it is only necessary when the parameters in the system have noticeable changes, for example, when a user tremendously changes the usage patterns such as the app change rate u. However, as the prior study on mobile usage shows users' maintain relatively consistent

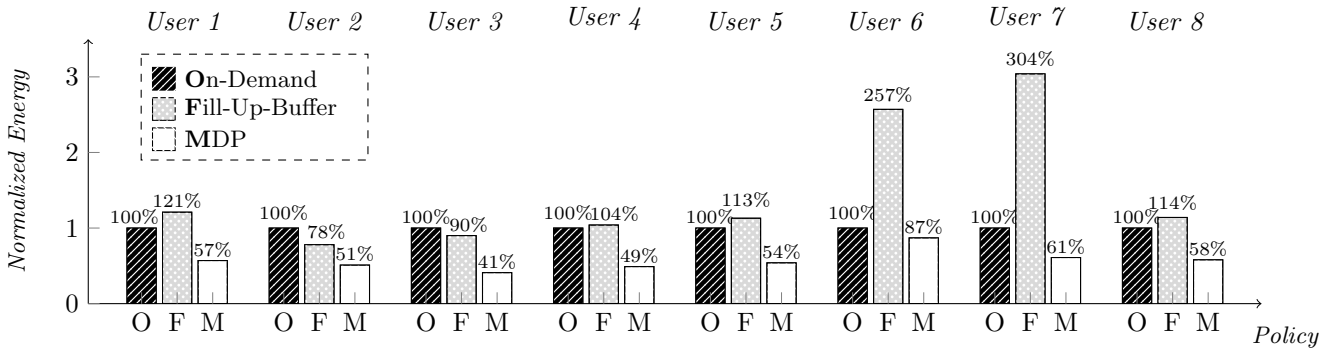


Fig. 6. Energy consumed by three ad-fetching policies when running 8 user traces. The energy consumption of all policies are normalized by that consumed by the default On-Demand Policy.

daily usage patterns, although the usage across different times of the same day may vary. This problem can be addressed to maintain certain user profiles in our Database module. Another system change that calls for re-optimization is when the user installs or uninstalls an ad-embedded app, however, for moderate amount of installations or uninstallations, the re-optimization workload can be safely postponed while still guaranteeing energy savings for other apps, for example, to the time when a user connects his/her device to a wall-charger. Besides, as designed in the algorithm, it is also possible to cache our MDP decision tables for given set of parameters, such that when similar usage patterns are recognized in the future, MDP optimization can be avoided to further reduce the computation complexity.

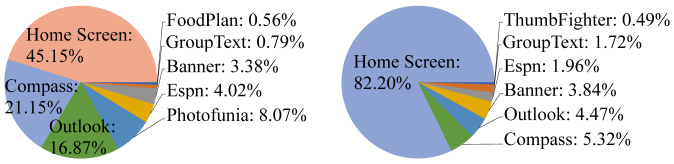
To measure the network-related energy, we build a software-based Energy Meter based on the approach adopted in prior work [31] and we tune the parameters for different network types and Internet service providers using measured energy traces from Monsoon Power Monitor [3]. With the timestamps recorded by our Proxy module, we are able to divide the overall energy consumption to ad-related traffic and other types of network traffic. The measurements and timestamps are summarized periodically and stored in our Database, to facilitate future analysis such as the parameter updating.

VI. Evaluation

A. Environment Setup

1) Hardware Setup: We utilize an external Monsoon Energy Monitor [3] to perform energy measurements and tune the parameters needed for software-based network energy measurement. All experiments are performed on Nexus 5.

2) Test User Traces: With our full system implementation, we adopt a trace-replay evaluation method to capitalize the effects of different ad prefetching policies in the real-world system environments. We collect 8 user traces from Rice LiveLab traces [26]. Each of the trace lasts for around 150 days. To guarantee comparison fairness, we randomly extract several periods of traces for each user and replay them on Android devices using the automated testing framework Monkey [5].



(a) User 2 (b) User 7

Fig. 7. App occupancies of two users.

B. Comparing EMAP against Baseline Policies

To guarantee a consistent evaluation environment, we replay the 8 collected user traces concurrently on three groups of phones. Each of the three policies is adopted by one of the three groups: On-Demand policy, heuristics-based Fill-Up-Buffer policy, and the MDP policy adopted by our EMAP system. As discussed in Section II-B, For On-Demand policy, all ads are fetched in real-time when an ad is requested. Fill-Up-Buffer policy maintains a fixed-size buffer and prefetches a batch of ads to fill up the available buffer, whenever ads are used up and a new ad is requested. Our policy makes the state-aware decisions in the Optimization Engine module by modeling the ad prefetching decisions as a Markov Decision Process. In this experiment, buffer size equals to 55 for FUB policy.

Figure 6 shows the ad-related energy comparison among the three policies. From the figure, we observe that our MDP policy always consumes less energy than the other two. For 6 out of 8 tested users, the saving exceeds 40%, and the maximum ad energy improvement upon the default On-Demand policy of contemporary ad libraries is 59%. The results in the figure show that our proposed state-aware MDP model in EMAP system is effective and consistent in energy saving. The Fill-Up-Buffer policy outperforms On-Demand for *User 2* and *User 3*, but is the worst for other users. In extreme cases (*User 6* and *User 7*), FUB is 157% and 204% worse than the default On-Demand policy.

We investigate the traces for representative *User 2* and *User 7*. By comparing the app occupancies of *User 2* (Figure 7(a)) and *User 7* (Figure 7(b)), we can see that "Home Screen" occupies much more percentage of time in the trace of *User 7* than that of *User 2*. "Home Screen" and Email app "Outlook" do not use any ad. Adding these two ad-free apps together, they consumed totally 86.67% of the experiment time of *User*

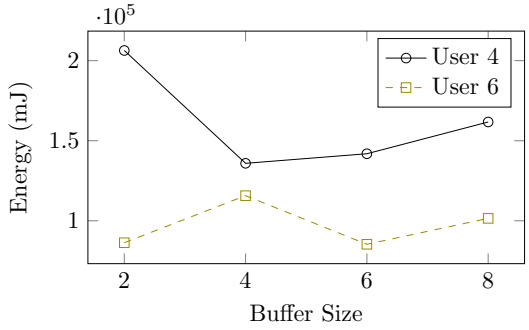


Fig. 8. Energy consumption for two user traces using Fill-Up-Buffer policy with different buffer size.

7, but 62.02% for *User 2*. Also, *User 2* runs the app "Espn" for a longer time, which has a very high ad display rate (8 ads/minute). During the experiments, apparently more ads are expected for *User 2* than *User 7*, thus prefetched ads are more likely to be used other than wasted.

For the On-Demand policy, the tail energy is a major drainer which impacts the performance badly. The heuristic Fill-Up-Buffer compresses the energy tails to the minimum amount, but due to its unawareness of system state changes, it fails to make agile decisions regarding the 'aggressiveness' in ad prefetching to adapt dynamically to runtime usage.

Our MDP policy seeks the sweet spot in the tradeoff between energy saving by ad prefetching and avoiding the ad expiration. In the following subsections, we will illustrate the decisions in details.

C. Studying Effect of Buffer Size on FUB Policy

From the previous result, we can see a fixed buffer size adopted by Fill-Up-Buffer policy fails to adapt to the runtime dynamics, which may be further affected by user behaviors. We further study the effect of buffer size for FUB policy in the current experiment. We use two users' traces and vary the buffer sizes and measure the energy consumption.

Figure 8 shows that for different users, the 'proper' buffer size for FUB policy may vary. Among the evaluated buffer sizes, while buffer size 4 works best for *User 4* (solid line), it is the worst for *User 6* (dashed line) and consumes the largest amount of energy.

On the other hand, from the energy consumptions for *User 4*, we can see as the buffer size increases, the energy consumption first decreases and then increases. However, such pattern is absent for *User 6*. This further confirms that simply fixing a single prefetching decision based on usage profile [20] is not effective to minimize ad-related energy.

In contrast, our formulation of EMAP problem in this paper jointly consider the system factors that have effects on optimal ads prefetching decisions: the ad display rates of active apps, network status and the number of residue ads. We will illustrate how the parameters we consider affect the optimal decisions made in our solution.

D. Studying Effect of Ad Display Rate

In this study, we first cumulatively measure the ad display rates for individual apps. Then we break down the prefetched ads when each individual app is active, and sum up the number of prefetched ads by our policy. The results are shown in Figure 9. The left and right y axises respectively denote the number of ads prefetched when the app is active, and the accumulated ad display rate for the app.

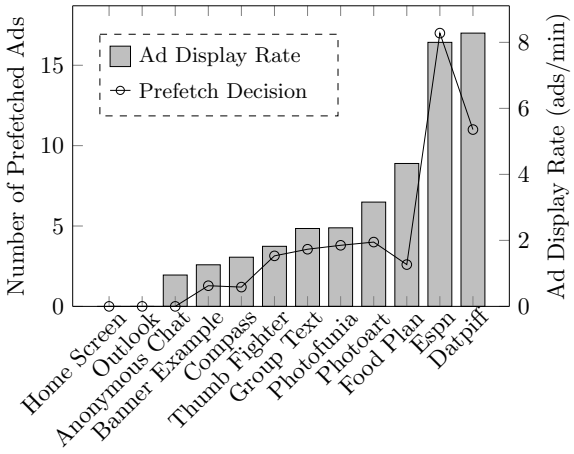


Fig. 9. The breakdown of average amount of prefetch ads when each single app is active on the foreground. The right y axis shows the average ad display rate for the app.

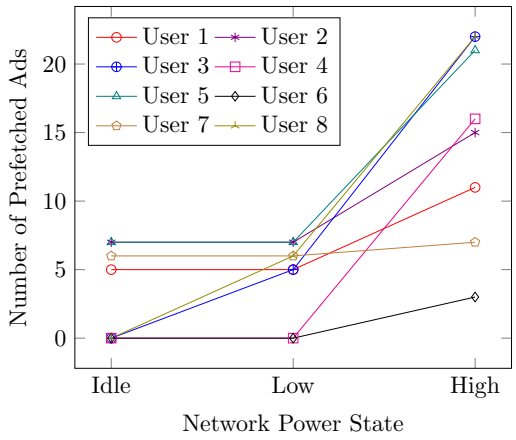


Fig. 10. Prefetch decisions under different network power states for all users

As seen from the figure, the amount of prefetching ads tends to increase as the ad display rates increase with apps "Food Plan" and "Datpiff" being the only exceptions. The reason is that both of these two apps consume much energy for non-ad data transmissions and keep the network status frequently in High Power state. In this case, the difference between the average cost generated by a prefetched ad and a possible future on-demand-fetched ad becomes smaller, and our cost minimization algorithm for the EMAP system considers the effects of future states. This is confirmed by the current set of result findings.

E. Studying Effects of Network Status

In the current set of experiment, we study the effect of another key parameter - current net status. For each individual user, we summarized the prefetching decisions made by our

algorithm when the network interfaces are working in different states.

As seen from the results shown in Figure 10, a common pattern across different users is that when network is in High Power state, it is preferred to prefetch more ads than in Low Power state, and the prefetching is least aggressive when network state is IDLE. This is because the larger amount of energy (with a full tail) will be generated when network state is IDLE, compared to no extra tail generated when network is in High Power state. Both current and previous experiments provide insights on how our proposed solution jointly consider different states in the system, and make the optimal decisions, leading to the huge energy savings, as shown in Figure 6.

VII. Related Work

Various techniques are proposed to save mobile energy in different problem domains, such as proxy-assisted browsing [27], content pre-staging [29], [12], [22], budget-based energy management [30], [10], wireless sensor networks [24], and study of app and user behaviors [6], [11], [25].

Existing work on mobile ad prefetching [20], [8] often adopts simple heuristics, yet makes strong assumptions such as single active app, fixed ad refreshing rate. Prior work [19], [28] also characterizes mobile advertisements on aspects like ad libraries, usage patterns, and ad deadlines. In contrast, our work is the first formal optimization framework on this problem and is evaluated on a full system implementation.

To minimize energy cost, Markov Decision Process has been previously applied to improve server sleep pattern [18], TCP connections [23], and email synchronization [9]. Our work comprehensively addresses the unique challenges in mobile ad optimization, considering ad expiration, system dynamics, and diversity in user/app behaviors. Another line of work on network measurement [17], [16], [21] also provides useful input to our work and other mobile energy optimization work [14], [22].

VIII. Conclusion

The tail states designed in cellular networks are intended to guarantee responsiveness for data connections. However, it causes severe energy inefficiency problem, especially for periodic and short data transmissions, such as those traffics made by contemporary ad libraries. Due to the deadlines imposed on ads, and also the runtime randomness resided in user behaviors, system states, traditional content prefetching fails to provide an effective solution for energy minimizing for ad-related traffic. We formulate the energy minimization problem for ad prefetching by modeling the decision making as a Markov Decision Process, considering important system states and events triggering state changes. Using our fully implemented system, we replay user traces to illustrate the effectiveness of our proposed solutions. The results demonstrate a consistent improvement in energy efficiency by our solution over existing On-Demand policy on Android by up to 59%, while a simple Fill-Up-Buffer policy can be even 3 times worse than the default On-Demand policy. Such findings capitalize on the promise of ad prefetching in real-world mobile systems.

References

[1] AdMob. https://developers.google.com/admob/.

[2] Livelab traces. http://livelab.recg.rice.edu/traces.html.

[3] Monsoon Powermeter. https://www.msoon.com/LabEquipment/PowerMonitor/.

[4] Optimizing downloads for efficient network access. https://developer.android.com/training/efficient-downloads/efficient-network-access.html.

[5] UI/Application exerciser monkey. http://developer.android.com/tools/help/monkey.html.

[6] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury. Detecting energy bugs and hotspots in mobile apps. In *SIGSOFT FSE*. ACM, 2014.

[7] N. Bäuerle and U. Rieder. *Markov decision processes with applications to finance*. Springer Science & Business Media, 2011.

[8] X. Chen, A. Jindal, and Y. C. Hu. How much energy can we save from prefetching ads?: Energy drain analysis of top 100 apps. In *HotPower*. ACM, 2013.

[9] T. L. Cheung, K. Okamoto, F. Maker III, X. Liu, and V. Akella. Markov decision process (mdp) framework for optimizing software on mobile phones. In *EMSOFT*. ACM, 2009.

[10] M. Dong, T. Lan, and L. Zhong. Rethink energy accounting with cooperative game theory. In *MobiCom*. ACM, 2014.

[11] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *MobiSys*. ACM, 2010.

[12] A. Finamore, M. Mellia, Z. Gilani, K. Papagiannaki, V. Erramilli, and Y. Grunenberger. Is there a case for mobile phone content pre-staging? In *CoNEXT*. ACM, 2013.

[13] A. Gerber, S. Sen, and O. Spatscheck. A call for more energy-efficient apps. *AT&T Labs Research*, 2011.

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

[15] R. A. Howard. Dynamic programming and markov processes.. 1960.

[16] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *MobiSys*. ACM, 2012.

[17] 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 *SIGCOMM*. ACM, 2013.

[18] I. Kamitsos, L. Andrew, H. Kim, and M. Chiang. Optimal sleep patterns for serving delay-tolerant jobs. In *e-Energy*. ACM, 2010.

[19] P. Mohan, S. Nath, and O. Riva. Prefetching mobile ads: Can advertising systems afford it? In *EuroSys*. ACM, 2013.

[20] P. Mohan, S. Nath, and O. Riva. Prefetching mobile ads: Can advertising systems afford it? In *EuroSys*. ACM, 2013.

[21] A. Nikravesh, H. Yao, S. Xu, D. Choffnes, and Z. M. Mao. Mobilyzer: An open platform for controllable mobile network measurements. In *MobiSys*. ACM, 2015.

[22] 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 *UbiComp*. ACM, 2013.

[23] C. Pluntke, L. Eggert, and N. Kiukkonen. Saving mobile device energy with multipath tcp. In *MobiArch*. ACM, 2011.

[24] C. Qiu, H. Shen, and K. Chen. An energy-efficient and distributed cooperation mechanism for k-coverage hole detection and healing in wsns. In *MASS*. IEEE, 2015.

[25] J. P. Rula, B. Jun, and F. Bustamante. Mobile ad (d): Estimating mobile app session times for better ads. In *HotMobile*. ACM, 2015.

[26] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum. LiveLab: measuring wireless networks and smartphone users in the field. In *ACM SIGMETRICS Performance Evaluation Review*, 2011.

[27] A. Sivakumar, S. Puzhavakath Narayanan, V. Gopalakrishnan, S. Lee, S. Rao, and S. Sen. PARCEL: Proxy assisted browsing in cellular networks for energy and latency reduction. In *CoNEXT*. ACM, 2014.

[28] A. Tongaonkar, S. Dai, A. Nucci, and D. Song. Understanding mobile app usage patterns using in-app advertisements. In *Passive and Active Measurement*. Springer, 2013.

[29] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. How far can client-only solutions go for mobile browser speed? In *WWW*. ACM.

[30] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *ASPLOS X*. ACM, 2002.

[31] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *CODES/ISSS*. ACM, 2010.