iMUTE: Energy-optimal Update Policy for Perishable Mobile Contents

Joohyun Lee[†], Fang Liu[†], Kyunghan Lee[‡], and Ness B. Shroff[♯],

***Abstract*—Mobile applications that provide ever-changing information such as social media and news feeds applications are designed to consistently update their contents in the background. This operation, often called "prefetching", provides the users with immediate access to up-to-date contents. However, such updates often result in the unwanted side-effect of draining the battery of mobile devices. It is considered as pure waste when updated contents are not accessed before being renewed. In this paper, we develop *an optimal strategy to update the contents in the background under a given energy constraint*. The key challenge is to predict when the user will access the contents in a probabilistic manner from the statistics of the accessed patterns in the past. We model our problem as a constrained Markov decision process (C-MDP) and propose to tackle its high complexity with a two-step solution that combines: (1) a threshold-based backward induction algorithm for the Lagrangian relaxation of our C-MDP, and (2) an iterative root finding algorithm, *iMUTE* (iterative Method for optimal UpdaTe policy with Energy constraint). We prove that *iMUTE* converges superlinearly to the optimal solution of the original C-MDP under a mild condition. We also experimentally verify that *iMUTE* outperforms the periodic policy as well as the additive and multiplicative increase policies that are adopted in the *Doze* mode of Android systems and HUSH, in terms of user experience and energy saving.**

I. Introduction

In the latest mobile devices, a myriad of user applications are designed to update their contents in the background to obtain better user experience. However, such updates are mostly unaware of the user behavior and often end up wasting the battery without effectively improving the user experience. Therefore, an immediate yet important question arises: *what is the optimal strategy to feed or update the contents in the background that keeps the contents as fresh as possible while meeting a given battery constraint?*

The energy wastage caused by inefficiently updating mobile application data has been reported in various studies such as [1]–[3]. Rosen et al. [1] revealed that many mobile applications such as social media, email, and information

[†]J. Lee and F. Liu are with the Department of Electrical and Computer Engineering at The Ohio State University, e-mails: lee.7119@osu.edu, liu.3977@buckeyemail.osu.edu.

[‡]K. Lee is with the School of Electrical and Computer Engineering, UNIST (Ulsan National Institute of Science and Technology), South Korea, e-mail: khlee@unist.ac.kr.

[♯]N. B. Shroff holds a joint appointment in both the Department of ECE and the Department of CSE at The Ohio State University, e-mail: shroff.11@osu.edu.

This research was supported in part by grants from the National Science Foundation CNS-1409336 and CNS-1446582, and the Army Research Lab MURI W911NF-12-1-0385, and the Army Research Office W911NF-14-1-0368. This research was supported in part by the MSIT(Ministry of Science and ICT), Korea, under the ITRC support program (IITP-2017-2017-0-01635) supervised by the IITP.

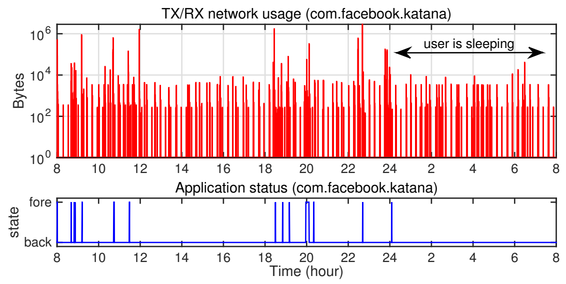


Fig. 1: Daily network usage and application states of the Facebook app. "fore" means foreground states (i.e., the user is interacting with the application), and "back" means background states. The intervals are less than 20 minutes.

delivery applications (e.g., weather and stocks) employ a naive periodic update policy and the background updates followed by tail energy[1] consume about 84% of the total communication energy. Another study by Pathek et al. [5] made a similar observation showing that 65% to 75% of the networking energy is spent on updating advertisements included in the applications in the background.

To more deeply understand this problem, we performed our own measurement with Galaxy S7 running Android 6.0.1, specifically for Facebook application, which is the most widely mentioned battery draining culprit. Our measurement for 24 hours captures intensive networking behaviors of the Facebook application as in Fig. 1, which persist even when the application is put in the background. Our energy measurement with the Monsoon power meter [6] further confirms that a single background update consumes about 1.7mAh (i.e., 6 Joule) and about 5% of the entire battery capacity is drained by the background updates of Facebook alone per day.

There are several practical works that are heuristically designed to alleviate the energy wastage from background activities. Android version 6.0 (Marshmallow), released in October 2015, adopts a new feature called *Doze* mode [7] for preventing an evident energy leakage. The *Doze* mode is enabled when a user leaves the device unattended for a certain amount of time (e.g., 2 hours), and increases the interval of restricting background behaviors (i.e., suppression interval), which in turn increases the update intervals of applications in the background. Another work [8] proposed a simple algorithm called HUSH that suppresses off-screen background activities. HUSH increases the suppression duration of a background

[1]Tail energy is the energy from being in the high power state after each data transfer during a predefined period (around 4 to 10 seconds). [4]

application exponentially[2] if it has not been brought to the foreground. Even with these simplistic approaches, it is shown that about 15-17% of the battery time can be extended.

We speculate that these algorithms are partially effective because it is typical that the users are less likely to launch an application as its unused period gets longer. This statistical behavior is explained by a so-called heavy-tailed inter-launching time distribution that is experimentally captured in [9], [10]. However, we point out that those algorithms neither perform closely to the optimal nor adapt flexibly to individual users.

In this work, we first formulate a contents update scheduling problem that minimizes the user inconvenience under an energy constraint.[3] Given the knowledge of usage statistics, we model the problem as a Constrained Markov decision process (**C-MDP**). To provide a solution, we take a divide and conquer approach. We first consider an Unconstrained MDP (**U-MDP**) and devise a low-complexity optimal backward induction algorithm by exploiting the threshold structure of optimal policies. Then, we develop an iterative algorithm based on the Dekker's method [12], in order to find a Lagrange multiplier by which the corresponding **U-MDP** becomes equivalent to the original **C-MDP** with a given constraint.

Our main contributions are as follows:

(1) We develop a threshold-based backward induction algorithm that solves **U-MDP** (i.e., minimizes a weighted sum of user inconvenience and energy cost).
(2) We develop *iMUTE*, an iterative algorithm based on the Dekker's method, which obtains an optimal Lagrange multiplier and an optimal update policy for a **C-MDP** problem. We prove that our iterative algorithm converges superlinearly under a mild condition.
(3) We report that the inter-launching times of social networking applications conform to a heavy-tailed distribution. Using trace-driven simulations over the real usage statistics of social applications, we demonstrate that our algorithm for **C-MDP** outperforms periodic and multiplicative/additive increase policies.

II. Related work

Suppression algorithms with increasing update intervals: Android version 6.0 (Marshmallow), released in October 2015, adopted a new feature called *Doze* mode [7] for energy saving. The *Doze* mode is enabled when a user leaves the device for a certain amount of time and restricts the background activities of applications. *Doze* mode has a short maintenance window during which the restrictions are relaxed, where the interval between maintenance windows increases as the device gets untouched longer. A recent paper [8] proposed a simple suppression algorithm called HUSH. HUSH increases the suppression interval of an application if it has not been used in the foreground using exponential backoff (i.e., the interval is multiplied by a scaling factor σ). Once an application is used in the foreground, the interval goes back to an initial value.

[2]Using an exponential backoff method in which the duration is multiplied by a given scaling factor.

[3]We consider a pull-based update system. A push-based update system (e.g., notification) is beyond the scope of this paper. We note that although push-based updates are highly responsive, they consume significant energy if such pushes happen frequently and incur user disruption [11].

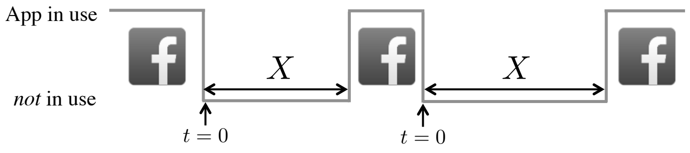
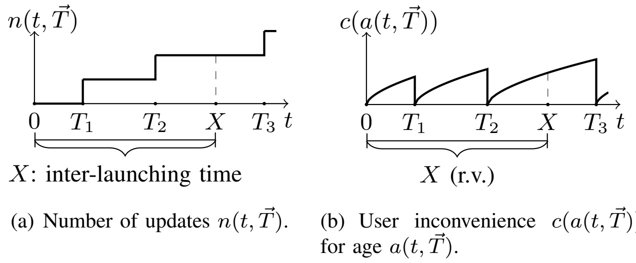


Fig. 2: Our system model for inter-launching time X (r.v.).



(a) Number of updates $n(t, \vec{T})$. (b) User inconvenience $c(a(t, \vec{T}))$ for age $a(t, \vec{T})$.

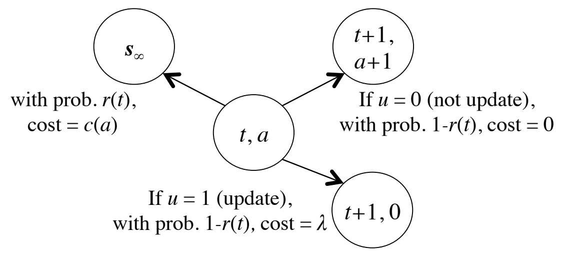
Fig. 3: Two types of costs (depicted with $c(a) = a^{0.6}$).

Inspection/maintenance problems: Inspection and maintenance problems [13]–[17] are closely related to our content update problem. The goal of an inspection problem is to efficiently detect the failure of a system, whose lifetime (i.e., inter-failure time) is arbitrarily distributed and the failure can be detected only by inspection. An optimal inspection policy minimizes the total expected cost composed of the inspection cost and the loss from the system failure. The main difference between an inspection problem and our problem is the cost. Our cost occurs from the time interval between an update and a system event (i.e., application launching) while the cost of the inspection problem comes from the time interval between a system event (i.e., system failure) and an inspection.

A maintenance problem is to find both inspection schedules and a replacement decision policy, where parts of machines deteriorate as they age. There are several approaches used to tackle these problems such as Markov renewal decision processes [15] and approximation algorithms from functional optimization under the assumption of concave cost functions [16]. However, most works have focused on solving a weighted sum cost minimization problem. To the best of our knowledge, there has been little effort in solving a constrained inspection or maintenance problem in which one of the costs is constrained. A constrained problem is more natural when the system has a budget for inspection/update, or targets to meet a specific system down/user inconvenience cost.

III. System model

We consider a time-slotted scheduling system for a mobile application that downloads data in order to update its contents in the background. Facebook, Twitter, and Gmail are examples of such applications. We assume that once a user stops using the application by putting it in the background, the user next uses this application after an inter-launching time X as depicted in Fig. 2, which we model as a discrete random variable with finite expectation $\mathbb{E}[X]$. The distribution of X



s_∞
$t+1, a+1$
with prob. $r(t)$, cost = $c(a)$
If u = 0 (not update), with prob. 1-$r(t)$, cost = 0
t, a
If u = 1 (update), with prob. 1-$r(t)$, cost = λ
$t+1, 0$

Fig. 4: State transition model with immediate costs in transition. State $\boldsymbol{s}$ is a tuple of the elapsed time t and age a.

is known a priori from history, and is stationary (i.e., the distribution does not change over time). X is also assumed to be independent from any update policy. This is a reasonable practical assumption because users in general revisit applications without the knowledge of how the background update works. We denote the elapsed time from the last user activity as a time slot t. During the inter-launching time, the update schedules $\vec{T} = (T_1, T_2, \cdots)$ specify when to retrieve contents from the server. We assume that the update completes instantly, because the sizes of update feeds are typically small.

We consider two types of costs, update cost $n(X, \vec{T})$ and user inconvenience $c(a(X, \vec{T}))$, where age $a(t, \vec{T}) = t - \max_{T_i \le t} T_i$, as depicted in Fig. 3. The age is the elapsed time from the latest background update. The update cost $n(t, \vec{T}) = \max\{i | T_i \le t\}$ is defined as the number of updates. We note that the energy consumed by updates is proportional to the number of updates.[4] The inconvenience cost captures the user's quality of experience degraded by *staleness* (i.e., *aging*) of contents. An older age for a certain application implies that there may exist a high amount of pending updates, hence results in longer latency in fetching all the latest feeds.[5] An application that has a large number of pending updates often slows the mobile device. The user inconvenience function $c(a)$ of age a is assumed to be non-decreasing in a, which is intuitive. This function can be of various forms with its dependency on the type of an application and user's attributes. Our problem is formally formulated to minimize the user inconvenience cost under a given update cost V as follows:

$$\textbf{P1:} \min_{\vec{T}} \mathbb{E}[c(a(X, \vec{T}))] \quad \text{subject to} \quad \mathbb{E}[n(X, \vec{T})] \le V. \quad (1)$$

The energy budget V can be tuned based on the battery level or target lifetime. The key challenge is that the update decision at a given timeslot is correlated with the costs of the entire future timeslots. To tackle the difficulty, we model it as a Markov decision process (MDP) with finite states and actions, which can capture these correlations with a value function of states.

A. Markov Decision Process

(1) State transition model: Our state transition model is illustrated in Fig. 4. The system *state* is defined as $\boldsymbol{s} = (t, a)$, a tuple of the elapsed time t and age $a \le t$. We also denote by $\boldsymbol{s}_t$ the system state at time slot t. The initial state $\boldsymbol{s}_0$ is $(0, 0)$. We let $t_{\max}$ be the maximum time slot, which can be regarded as the longest time the system tracks, and $t \le t_{\max}$. In practice, $t_{\max}$ is typically no longer than 24 hours for actively used applications. We also define a terminating (i.e., absorbing) state $\boldsymbol{s}_\infty$ in which the system state enters whenever the user launches the application. We let S denote the set of all states, and $S' = S \setminus \boldsymbol{s}_\infty$ denote the transient states.

To model the transition to a terminating state, we define the failure rate $r_X(t)$ that quantifies the frequency that a random variable X is t (e.g., the user launches the application at t) under the condition that $X \ge t$ [18]. We omit the subscript X and use $r(t) = r_X(t)$ unless confusion arises. Formally, $r_X(t) \triangleq \frac{f_X(t)}{1 - F_X(t)}$, for t such that $F_X(t) < 1$, where $f_X(t)$ and $F_X(t) = \mathbb{P}[X \le t]$ are the probability mass function and cumulative distribution function of X, respectively. X is said to have *negative (positive) aging* [16] when its failure rate strictly decreases (increases) as time goes by. Intuitively, the failure rate $r_X(t)$ defines the probability that the system goes to the terminating state at the elapsed time t. Typical machines go through positive aging as their failing probabilities increase as time goes by. We define $\mu : \boldsymbol{s} \to \mu(\boldsymbol{s})$ as an update policy which is defined for each state $\boldsymbol{s} = (t, a)$, where a control $\mu(\boldsymbol{s}) = \mu(t, a) \in \{0, 1\}$ decides whether to update (=1) or not to update (=0) contents at each state $\boldsymbol{s}$. In this paper, we only consider the class of deterministic stationary policies, since there exists an optimal deterministic stationary policy in a transient Markov decision process with non-negative immediate costs from Theorem 9.1 in [19].

We define the transition probability from state $\boldsymbol{s} = (t, a)$ to state $\boldsymbol{s}' = (t', a')$ under a control u by $p(\boldsymbol{s}' | \boldsymbol{s}, u) = p((t', a') | (t, a), u)$. Note that $\sum_{\forall \boldsymbol{s}' \in S} p(\boldsymbol{s}' | \boldsymbol{s}, u) = 1, \forall \boldsymbol{s}, u$. For the transition from other states to the terminating state $\boldsymbol{s}_\infty$, we have

$$p(\boldsymbol{s}_\infty | (t, a), u) = r(t), \forall u \in \{0, 1\}.$$

If the next state $\boldsymbol{s}'$ is not the terminating state, s_∞,

$$p((t', 0) | (t, a), 1) = 1 - r(t), \; p((t', a+1) | (t, a), 0) = 1 - r(t),$$

for $t' = \min(t + 1, t_{\max})$. If $\mu(s) = 1$, the age is reset to 0 as the device refreshes the contents, and if $\mu(s) = 0$, the age and the elapsed time slot are increased by one time slot. In all other cases, the transition probability is zero.

(2) Cost model for MDP: Immediate update and inconvenience costs in state transition from $\boldsymbol{s} = (t, a)$ to $\boldsymbol{s}' = (t', a')$ under a control u are denoted by $g_n(\boldsymbol{s}, u) = g_n((t, a), u)$ and $g_c(\boldsymbol{s}' | \boldsymbol{s}, u) = g_c((t', a') | (t, a), u)$, respectively. Note that the update cost is not dependent on the next state.

$$g_n(\boldsymbol{s}, u) = \mathbf{1}_{\{u=1\}}, \; g_c(\boldsymbol{s}' | (t, a), u) = c(a) \cdot \mathbf{1}_{\{\boldsymbol{s}' = \boldsymbol{s}_\infty, u=0\}},$$

where $\mathbf{1}_{\{\cdot\}}$ is the indicator function. The expected update and inconvenience costs from a scheduling policy μ are denoted by $N(\mu)$ and $C(\mu)$, respectively:

$$N(\mu) = \mathbb{E}\Big[\sum_{t=0}^{X} g_n(\boldsymbol{s}_t, \mu(\boldsymbol{s}_t))\Big], C(\mu) = \mathbb{E}\Big[\sum_{t=0}^{X} g_c(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, \mu(\boldsymbol{s}_t))\Big].$$

[4]This assumption is valid when the amount of download data in an update is relatively small and the update interval is longer than a tail duration (typically about 10 seconds) as most energy consumption is from the tail energy for small-sized packets [4].

[5]Long start-up latency of applications is one of the major complaints of smartphone users in a user survey of about 100 users [10].

Remark 3.1: The Markov decision process (S, p, μ) is a S'-transient MDP or absorbing to $\boldsymbol{s}_\infty$ for all policies.

(3) Problem Definition: Our objective is to minimize the *expected user inconvenience* $C(\mu)$ under a given constraint on the *expected number of updates*, $N(\mu) \leq V$, where V is the threshold. We define the **C-MDP** (Constrained MDP) which is equivalent to **P1** as follows:

$$\textbf{C-MDP:} \quad \min_{\mu \in U} C(\mu) \quad \text{subject to} \quad N(\mu) \leq V. \tag{2}$$

It is well known that a **C-MDP** problem is hard to solve by using standard MDP solution techniques such as dynamic programming. Thus, we define an **U-MDP(λ)** (Unconstrained MDP) problem with a Lagrange multiplier $\lambda > 0$, as follows:

$$\textbf{U-MDP(}\lambda\textbf{):} \quad \min_{\mu \in U} C(\mu) + \lambda \cdot (N(\mu) - V). \tag{3}$$

From the Theorem 9.9 in [19], it is proven that solving **C-MDP** is equivalent to solving the corresponding **U-MDP(λ)** for some λ under the transient framework and non-negative immediate costs. Our Markov decision process model satisfies these conditions. In **U-MDP(λ)**, we denote μ^*_λ as an optimal solution of **U-MDP(λ)**, i.e., $\mu^*_\lambda = \arg\min_{\mu \in U} C(\mu) + \lambda \cdot N(\mu)$. Note that V is a constant term in **U-MDP(λ)**. We define $C^o(\lambda) = C(\mu^*_\lambda)$ and $N^o(\lambda) = N(\mu^*_\lambda)$ to emphasize the dependency of optimal expected costs on λ.

Proposition 3.1: (i) $N^o(\lambda)$ is non-increasing and (ii) $C^o(\lambda)$ is non-decreasing in λ.

Proof: We prove (i) and (ii) by contradiction. Consider any λ_1 and λ_2 such that $\lambda_1 < \lambda_2$.
(i) Suppose $N^o(\lambda_2) > N^o(\lambda_1)$. From the optimality of $\mu^*_{\lambda_2}$,

$$C^o(\lambda_2) + \lambda_2 N^o(\lambda_2) \leq C^o(\lambda_1) + \lambda_2 N^o(\lambda_1).$$

By rearranging terms and using $\lambda_2 > \lambda_1$,

$$C^o(\lambda_1) - C^o(\lambda_2) \geq \lambda_2(N^o(\lambda_2) - N^o(\lambda_1)) > \lambda_1(N^o(\lambda_2) - N^o(\lambda_1)).$$

Thus, $C^o(\lambda_2) + \lambda_1 N^o(\lambda_2) < C^o(\lambda_1) + \lambda_1 N^o(\lambda_1)$ and it contradicts that $\mu^*_{\lambda_1}$ is optimal for **U-MDP(λ_1)**.
(ii) Suppose that $C^o(\lambda_2) < C^o(\lambda_1)$. Then, from (i),

$$C^o(\lambda_2) + \lambda_1 N^o(\lambda_2) < C^o(\lambda_1) + \lambda_1 N^o(\lambda_1).$$

It contradicts that $\mu^*_{\lambda_1}$ is optimal for **U-MDP(λ_1)**. ■

We denote λ^*_V as the optimal Lagrange multiplier such that **C-MDP** is equivalent to **U-MDP(λ^*_V)** for the threshold V. Deriving the optimal Lagrange multiplier for a threshold V is not trivial since we do not have any explicit expressions for $N^o(\lambda)$ or $C^o(\lambda)$. Thus, the proposed algorithms to solve the **C-MDP** problem are divided into two parts. We first focus on solving **U-MDP(λ)** for a Lagrange multiplier λ. Then, we develop an iterative algorithm based on the Dekker's method [12], to find the optimal Lagrange multiplier λ^*_V for the equivalent **U-MDP(λ)** problem to **C-MDP** problem with a threshold V.

IV. Algorithm Design

In this section, we propose *iMUTE* (iterative Mechanism for optimal UpdaTe policy with Energy constraint) to solve **C-MDP**. *iMUTE* consists of two stages: (A) an optimal backward induction algorithm for **U-MDP(λ)** and (B) an iterative algorithm to find λ^*_V. We explain these stages in each subsection.

A. *Optimal Update Policy for* U-MDP(λ)

For a fixed λ, there exists an optimal stationary deterministic policy for **U-MDP(λ)** from Theorem 9.1 in [19]. The policy iteration algorithm [20] updates a value and a control of each state iteratively to find an optimal policy, where a *value function* $h^{(k)}(\boldsymbol{s})$ is defined as the expected sum of the immediate costs g from state $\boldsymbol{s}$ to the terminating state $\boldsymbol{s}_\infty$, at k-th iteration. $g(\boldsymbol{s}'|\boldsymbol{s}, u)$, the immediate cost at a state $\boldsymbol{s}$ for a control u, is defined as the weighted sum of immediate update and inconvenience costs, i.e., $g(\boldsymbol{s}'|\boldsymbol{s}, u) = g_c(\boldsymbol{s}'|\boldsymbol{s}, u) + \lambda \cdot g_n(\boldsymbol{s}, u)$. The initial value function, $h^{(0)}(\boldsymbol{s})$ can be set arbitrarily (e.g., $h^0(\boldsymbol{s}) = 0$) for all state $\boldsymbol{s}$. The policy iteration algorithm that converges to an optimal policy μ^*_λ for **U-MDP(λ)** is as follows [20]:

$$\mu^{(k+1)}(\boldsymbol{s}) = \operatorname*{argmin}_{u \in \{0,1\}} \hat{h}^{(k)}(\boldsymbol{s}, u),$$
$$h^{(k+1)}(\boldsymbol{s}) = \hat{h}^{(k+1)}(\boldsymbol{s}, \mu^{(k+1)}(\boldsymbol{s})),$$

where $\hat{h}^{(k+1)}(\boldsymbol{s}, u) = \sum_{\boldsymbol{s}'} p(\boldsymbol{s}'|\boldsymbol{s}, u) \left[h^{(k)}(\boldsymbol{s}') + g(\boldsymbol{s}'|\boldsymbol{s}, u)\right]$. When $\max_{\boldsymbol{s}} |h^{(k+1)}(\boldsymbol{s}) - h^{(k)}(\boldsymbol{s})| < \eta, \forall s$, for a sufficiently small η, the policy $\mu^* = \mu^{(k+1)}$ becomes an optimal scheduling policy. We also use $h(\boldsymbol{s})$ as the value of state $\boldsymbol{s}$ after they are converged. It is well known that this optimal algorithm is practically hard to use due to the curse of dimensionality [21]. The computational complexity involved is quadratic in the number of states, where the number of states is also quadratic in the maximum time slot, i.e., $t_{\max}$. Therefore, the complexity is $O(t^4_{\max})$. To reduce the complexity, we first prove that a threshold-based policy is optimal, and then develop an optimal backward induction algorithm, whose complexity is $O(t^2_{\max})$.

Theorem 4.1 (Threshold structure of an optimal policy):

(i) The value function $h(t, a)$ is non-decreasing in a, and
(ii) an optimal policy $\mu^*(t, a)$ is non-decreasing in a. In other words, an optimal policy updates contents if and only if $a \geq a^*(t)$ at time t for the *threshold age* $a^*(t)$ at t.

Proof: We prove (i) and (ii) by induction. For an update policy $u \in \{0, 1\}$, $\hat{h}^{(k+1)}(t, a, u)$ is computed as follows.

$$\hat{h}^{(k+1)}(t, a, 1) = \lambda + (1 - r(t))h^{(k)}(t', 0),$$
$$\hat{h}^{(k+1)}(t, a, 0) = r(t)c(a) + (1 - r(t))h^{(k)}(t', a + 1),$$

where $t' = \min(t+1, t_{\max})$. Initially, set $h^{(0)}(t, a) = 0$. At the first iteration, $\mu^{(1)}(t, a) = 1$ if $r(t)c(a) > \lambda$ and 0 otherwise. Since $c(a)$ is non-decreasing in a, $\mu^{(1)}(t, a)$ is non-decreasing in a for all t. Also, $h^{(1)}(t, a) = \min(\lambda, r(t)c(a))$ which is non-decreasing in a.

Suppose that $h^{(k)}(t, a)$ and $\mu^{(k)}(t, a)$ are non-decreasing in a. We will show that $h^{(k+1)}(t, a)$ and $\mu^{(k+1)}(t, a)$ are non-decreasing in a. Observe that $\hat{h}^{(k+1)}(t,a,1)$ is not dependent on a. $\hat{h}^{(k+1)}(t,a,0)$ is non-decreasing in a, since both $c(a)$ and $h^{(k)}(t',a+1)$ are non-decreasing in a. Therefore, if $\hat{h}^{(k+1)}(t,a,0) > h^{(k+1)}(t,a,1)$ for some a, then $\hat{h}^{(k+1)}(t,a+1,0) > \hat{h}^{(k+1)}(t,a+1,1)$. This indicates that $\mu^{(k+1)}(a) = 1 \rightarrow \mu^{(k+1)}(a+1) = 1$, i.e., $\mu^{(k+1)}(a)$ is non-decreasing in a. Also, $h^{(k+1)}(t,a)$ is non-decreasing in a from $h^{(k+1)}(t,a) = \min(\hat{h}^{(k+1)}(t,a,1), \hat{h}^{(k+1)}(t,a,0))$. ■

Before we describe our threshold-based backward induction algorithm, we first consider the case with geometrically distributed X. Because of its memoryless property, we can simply calculate the optimal threshold age and this will be used in calculating the optimal threshold age at the maximum time slot, $t_{\max}$ in the general distribution.

(1) Geometrically distributed X (constant failure rate): If the inter-launching time X is memoryless or geometrically distributed (i.e., the failure rate is constant, or $r(t) = r$), then we can ignore the state variable t. Thus, we can simply use a to denote the state and $\hat{h}(a;\mu)$ as the value of the state a with update policy μ. In such a case, we can find an optimal threshold age using a simple equation, and then a periodic update policy with the chosen threshold age becomes optimal. Denote $\mu_{a'}$ as an update policy such that $\mu_{a'}(a) = 1$ for $a \geq a'$ and 0 otherwise. For a threshold age a', the value function at the initial state $a = 0$ is computed as follows.

$$\hat{h}(0;\mu_{a'}) = \frac{\lambda(1-r)^{a'+1} + \sum_{i=1}^{a'-1} r(1-r)^i c(i)}{1-(1-r)^{a'+1}}, \quad (4)$$

from $\hat{h}(a;\mu_{a'}) = \lambda + (1-r)\hat{h}(0;\mu_{a'})$ for $a \geq a'$ and $\hat{h}(a;\mu_{a'}) = r \cdot c(a) + (1-r)\hat{h}(a+1;\mu_{a'})$ for $a < a'$. Then, the optimal threshold age is $a^* = \operatorname{argmin}_a \hat{h}(0;\mu_a)$. If multiple threshold ages attain minimum, we choose one randomly.

(2) Threshold-based optimal update policy for U-MDP(λ): Using the Theorem 4.1 and the results from a geometrically distributed inter-launching time, we develop a threshold-based backward induction algorithm that obtains an optimal update policy for **U-MDP(λ)**. Eq. (4) is used to obtain an optimal threshold age at the maximum timeslot ($t_{\max}$) in step **(A)** and Theorem 4.1 guarantees the optimality of the threshold age obtained in backward induction in step **(B)**. The formal algorithm description is as follows.

***Threshold-based backward induction for* U-MDP(λ)**

Inputs: $\lambda, t_{\max}, r(\cdot), c(\cdot)$
Outputs: $\mu^*, N(\mu^*)$

```
(A) Computing $a^*(t_{\max})$:
1: $t = t_{\max}, r = r(t_{\max})$.
2: $a^*(t) = \operatorname{argmin}_a \frac{\lambda(1-r)^{a+1}+\sum_{i=1}^{a-1} r(1-r)^i c(i)}{1-(1-r)^{a+1}}$.
3: $h(t,0) = \frac{\lambda(1-r)^{a^*(t)+1}+\sum_{i=1}^{a^*(t)-1} r(1-r)^i c(i)}{1-(1-r)^{a^*(t)+1}}$.
4: $h_n(t,0) = \frac{\lambda(1-r)^{a^*(t)+1}}{1-(1-r)^{a^*(t)+1}}$.
5: $h(t,a) = \lambda + (1-r)h(t,0)$ for $a \geq a^*(t)$.
6: $h(t,a) = r \cdot c(a) + (1-r)h(t,a+1)$ for $1 \leq a < a^*(t)$.
7: $h_n(t,a) = \mathbf{1}_{\{a \geq a^*(t)\}} + (1-r(t))h_n(t,0)$, for $a \geq 1$.
(B) Backward induction:
8: for $t = t_{\max}-1$, $t = 0$, $t--$ do
9:     if $\hat{h}(t,a^*(t+1),1) < \hat{h}(t,a^*(t+1),0)$,     ▷ $a^*(t) \leq a^*(t+1)$
10:        for $a = a^*(t+1)-1$, $a = 1$, $a--$ do
11:            if $\hat{h}(t,a,1) \geq \hat{h}(t,a,0)$,
12:                $a^*(t) = a+1$; break
13:    else     ▷ $a^*(t) > a^*(t+1)$
14:        for $a = a^*(t+1)+1$, $a = t$, $a++$ do
15:            if $\hat{h}(t,a,1) < \hat{h}(t,a,0)$,
16:                $a^*(t) = a$; break
17:    $\mu^*(t,a) = 1$ for $a \geq a^*(t)$ and 0 otherwise.
18:    $h(t,a) = \hat{h}(t,a,\mu^*(t,a))$.
19:    $h_n(t,a) = \mathbf{1}_{\{a \geq a^*(t)\}} + (1-r(t))h_n(t+1,0)$.
20: $N(\mu^*) = h_n(0,0)$.
```

In line 2, if there are multiple minima, it chooses one randomly to break the tie. In Step **(B)**,

$$\begin{aligned}\hat{h}(t,a,0) &= r(t)c(a) + (1-r(t))h(t+1,a+1),\\ \hat{h}(t,a,1) &= \lambda + (1-r(t))h(t+1,0).\end{aligned}$$

$h_n(t,a)$ is the value function for the update cost, which accumulates immediate update cost $g_n(\boldsymbol{s},\mu^*(\boldsymbol{s}))$ in the future time slots. In each time slot t of backward induction, we first check whether $a^*(t)$ is smaller than or equal to $a^*(t+1)$, or larger than $a^*(t+1)$ in line 9. Note that if $\hat{h}(t,a^*(t+1),1) < \hat{h}(t,a^*(t+1),0)$, then an optimal policy should update at state $(t,a^*(t+1))$, i.e., $a^*(t) \leq a^*(t+1)$. The intuition behind this is that if $r(t)$ is slowly varying in t such that $r(t) \sim r(t')$ for $t < t' < t + a^*(t+1)$, then both the immediate costs and next value functions at states $(t,a^*(t+1))$ and $(t+1,a^*(t+1))$ are similar so that $a^*(t)$ is also similar to $a^*(t+1)$. If $r(t)$ is slowly varying, when the failure rate X is decreasing (increasing) or heavy-tailed, the optimal threshold age is non-decreasing (non-increasing) in t, i.e., the update intervals are increasing as the elapsed time t increases. We will show this in Section V. This coincides with the approximation algorithm in [16]. After checking this, depending on the cases, $a^*(t) \leq a^*(t+1)$ or $a^*(t) > a^*(t+1)$, the algorithm searches for the optimal threshold age at time t by decreasing age in lines 10-12 or by increasing age in lines 14-16, respectively. The value functions h and h_n are updated in lines 3-7 and 18-19. Finally, the expected update cost at the initial state $(0,0)$, $N(\mu^*)$ is equal to $h_n(0,0)$.

B. A Dekker's method-based iterative algorithm to find the optimal Lagrange multiplier λ_V^*

Now, the next problem is to find a solution of $g(\lambda) = N(\lambda) - V = 0$. Since $N(\lambda)$ is non-increasing in λ and $N(\lambda) \geq 0$, there exists a solution for $0 < V \leq N(0)$ by the intermediate value theorem. This problem is also equivalent to a minimization problem, $\min(g(\lambda))^2$. We note that the maximum of $N(\lambda)$ is $N(0) = \mathbb{E}[X]$, where the corresponding optimal scheduling policy is to update contents at every time slot. The solution can be found by iterative algorithms. A desired property is fast convergence, since an evaluation of $N(\lambda)$ requires solving the **U-MDP**(λ) at each iteration.

A reliable way is to use the bisection method, whose rate of convergence[6] is $\frac{1}{2}$. In order to achieve much faster convergence, we adopt a hybrid approach, called Dekker's method [12], which is a combination of the bisection method and the secant method.[7] The Dekker's method is as reliable as the bisection method, and as fast as the secant method when the function is "*well-behaved*", which we will specify later. The *secant method* is an approximation of Newton's method, which is useful when the derivative function g' is not available, as in our problem. Instead of using the derivative, the secant method approximates the derivative from the function values at the last two iteration points. From this, the secant method is also called linear interpolation. The rate of convergence of the secant method is 1.62, which is superlinear.

We first describe our iterative algorithm based on Dekker's method, and provide the fast convergence result of our algorithm, i.e., the secant step is performed at each iteration, under a condition that the time slot is sufficiently short.

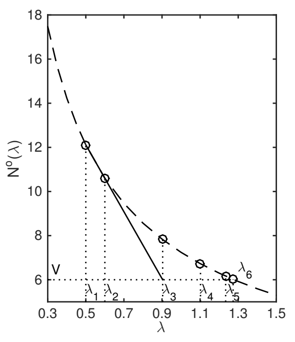
iMUTE: Dekker's-method-based iterative algorithm

```
Inputs: $V$ (threshold), $\epsilon$ (tolerance)
Outputs: $\lambda_V^*$, $\mu^*_{\lambda_V^*}$
(A) Initialization:
1: Choose sufficiently small $\lambda_1, \lambda_2 > 0$ ($\lambda_1 < \lambda_2$), and large
   $\lambda_m$ such that $N^o(\lambda_2) > V$ and $N^o(\lambda_m) < V$.
2: $N^o(\lambda_1) \leftarrow$ U-MDP$(\lambda_1)$, $N^o(\lambda_2) \leftarrow$ U-MDP$(\lambda_2)$, $k = 2$.
(B) Iteration step:
3: while $|N^o(\lambda_k) - V| \leq \epsilon$
4:     if $N^o(\lambda_k) \neq N^o(\lambda_{k-1})$,
5:         $\lambda_{k+1} = \lambda_k - \frac{\lambda_k - \lambda_{k-1}}{N^o(\lambda_k) - N^o(\lambda_{k-1})} \cdot (N^o(\lambda_k) - V)$.
6:     else $\lambda_{k+1} = \frac{\lambda_k + \lambda_m}{2}$.
7:     $N^o(\lambda_{k+1}), \mu \leftarrow$ U-MDP$(\lambda_{k+1})$.
8:     if $sign(N^o(\lambda_{k+1}) - V) = sign(N^o(\lambda_m) - V)$, $\lambda_m = \lambda_k$.
9:     $k = k + 1$.
10: $\lambda_V^* = \lambda_k$, $\mu^*_{\lambda_V^*} = \mu$.
```

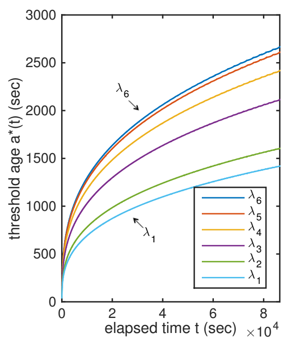
Initially, we choose sufficiently small λ_1, λ_2 such that $N^o(\lambda_2) > N^o(\lambda_1) > V$, and large λ_m such that $N^o(\lambda_m) < V$. From the monotonicity of $N^o(\lambda)$, the optimal λ^* lies in $[\lambda_2, \lambda_m]$. λ_m will be updated such that the optimal λ^* is between λ_k and λ_m. In the iteration, the secant step in line 5 is performed unless $N^o(\lambda_k) = N^o(\lambda_{k-1})$, as illustrated numerically in Fig. 5(a). The condition $N^o(\lambda_k) = N^o(\lambda_{k-1})$ is satisfied when the difference in λ_k and λ_{k-1} becomes sufficiently small so that the update intervals are not changed. We note that the intervals are discrete and this may happen

[6]The rate of convergence is $\lim_{k\to\infty} \frac{\lambda_{k+1} - \lambda^*}{\lambda_k - \lambda^*}$, where λ_k is the value at k-th iteration.

[7]It is also called Broyden's method [22] for multiple variables.



(a) Dekker's-method-based iterations (from the secant steps).



(b) Optimal threshold age.

Fig. 5: Numerical results of our iteration algorithm and the optimal threshold age in each iteration. The chosen parameters are $\mathbb{E}[X] = 7200$ (sec), $\beta = 0.5$, $V = 6$, $c(a) = a^{0.5}$, and $\lambda_1 = 0.5$, $\lambda_2 = 0.6$. The dashed line indicates the expected update cost $N^o(\lambda)$ and the solid line indicates the linear interpolation (the secant step) at the first iteration. The white dots indicate iteration points $(\lambda_k, N^o(\lambda_k))$.

when λ_k approaches close to the optimal λ^*. Then, the bisection step in line 6 is used. After the update of λ_k, depending on the signs of $N^o(\lambda_{k+1}) - V$ and $N^o(\lambda_m) - V$, λ^* can be in either $[\lambda_k, \lambda_{k+1}]$ or $[\lambda_{k+1}, \lambda_m]$ when $\lambda_k < \lambda_m$.[8] If their signs are the same, λ^* is in $[\lambda_k, \lambda_{k+1}]$, and we update λ_m to be λ_k, as in line 8. Otherwise, λ^* is in $[\lambda_{k+1}, \lambda_m]$. This bisection interval guarantees the convergence from the monotonicity of $N^o(\lambda)$.

Superlinear convergence of our iterative algorithm: The secant method is proven to superlinearly converge when the function $N^o(\lambda)$ is Lipschitz continuous [23].[9] Thus, if the derivative of $N^o(\lambda)$ is bounded, our Dekker's method converges superlinearly by the secant step. We prove the following lemma for the bounded derivative of $N^o(\lambda)$ in Appendix.

Lemma 4.1 (Bounded derivative of $N^o(\lambda)$): If the length of a time slot becomes infinitely small, the derivative of $N^o(\lambda)$ is bounded.

If the time slot is not sufficiently short, the derivative may not be bounded. Even in this case, the bisection step in our iterative algorithm guarantees the convergence and the rate of convergence is $\frac{1}{2}$.

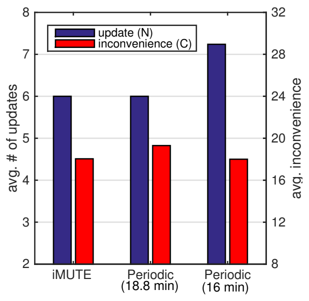
V. Numerical Results

In this section, we illustrate our iteration algorithm and the threshold-based update policy for Weibull distributed (heavy-tailed) inter-launching time X.[10] Later, we will show that the empirical inter-launching times of users follow the Weibull

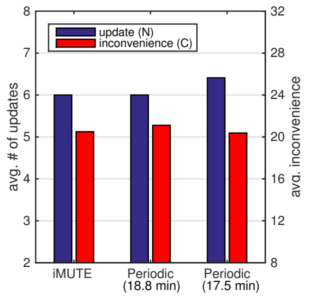
[8]When $\lambda_k > \lambda_m$, λ^* can be in either $[\lambda_m, \lambda_{k+1}]$ or $[\lambda_{k+1}, \lambda_k]$.

[9]A function $f : X \to Y$ is called Lipschitz continuous if there exists a real constant $K \geq 0$ such that for all $x_1, x_2 \in X$, $|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$.

[10]The probability mass function of the discrete Weibull distribution with the parameters α (scale) and β (shape) is $\exp[-(\frac{t}{\alpha})^\beta] - \exp[-(\frac{t+1}{\alpha})^\beta]$. When $\beta < 1$, the distribution is negative aging or heavy-tailed, i.e., $r_X(t)$ is decreasing in t.



(a) $\beta = 0.5$, more heavy-tailed.



(b) $\beta = 0.7$, less heavy-tailed.

Fig. 6: Expected update and inconvenience costs of *iMUTE* and the periodic policy, with different shape parameters β. The numbers in the parenthesis indicate the update period of the periodic policy. When either of update or inconvenience cost is equally adjusted, the other cost is always less in *iMUTE* compared to the periodic policy.

distribution in our traces. We use the following parameters: $\mathbb{E}[X] = 7200$ (sec), $\beta = 0.5$, $V = 6$, $c(a) = a^{0.5}$, and $\lambda_1 = 0.5$, $\lambda_2 = 0.6$. The length of a time slot is 1 sec. We depict the results in Fig. 5. Fig. 5(a) illustrates the Dekker's-method-based iteration algorithm. In every iteration, the secant step is performed and the algorithm converges to the optimum in 4 steps with $\epsilon = 0.02$. At the optimal Lagrange multiplier $\lambda_V^* = \lambda_6$, we obtain the optimal update policy for **C-MDP** as shown in Fig. 5(b). Because the failure rate X is decreasing, the optimal threshold age is non-decreasing in t, i.e., the update intervals are increasing as the elapsed time t increases.

In Fig. 6, we compare *iMUTE* with periodic policies with the same update cost and the same user inconvenience cost, for two different shape parameters, β of the Weibull distribution. Note that as the shape parameter decreases, the distribution becomes more heavy-tailed. The interval of a periodic policy that has the same update cost has a slightly lower interval than $\mathbb{E}[X]/V = 20$ minutes, since the update cost is counted after each update interval. For the shape parameter $\beta = 0.5$, our optimal update policy decreases the update cost by 21% and the user inconvenience cost by 7%. For the shape parameter $\beta = 0.7$, the gains become 7% for the update cost and 3% for the user inconvenience cost. Thus, our threshold-based policy becomes more effective compared to the periodic policy, as the inter-launching time distribution becomes more heavy-tailed.

VI. Trace-driven simulation

In this section, we evaluate the performance of our algorithm over real smartphone traces by comparing with other benchmark schemes. The key performance metrics are the update cost and user inconvenience from stale contents.

A. Traces: Inter-launching time

To capture application usage behaviors of smartphones in the wild, we performed our own data collection with 101 Android users selected from a few popular Internet communities of South Korea two times during Feb. 5-18, 2015 and July 7-20, 2016. In order not to incur any bias, we did not inform that the social media applications are of our particular interests. The most popular social application in our trace data is Facebook followed by Naver Cafe (a community app) and Naver Band (a group chat app). We focus on Facebook in our evaluation, where the simulation results are similar in any social media application that is of main use. For more details, please refer to [30]. For the trace-driven simulations, we select 26 users who regularly use Facebook at least once a day.

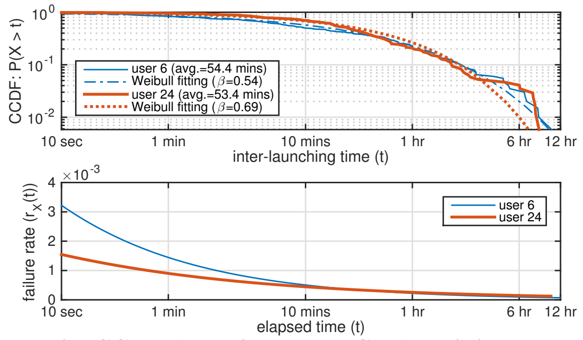


Fig. 7: The CCDF (complementary CDF) and the corresponding Weibull fitting of the inter-launching times (top) and the failure rates estimated from the fitting (bottom) of two users.

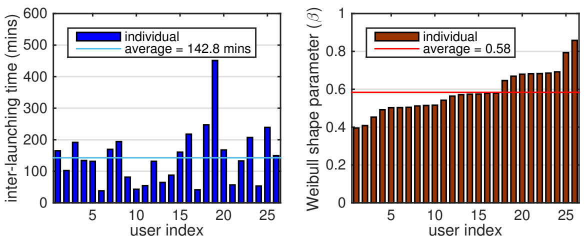


Fig. 8: The average inter-launching times (left) and the best fitted Weibull shape (β) parameters (right) of the users.

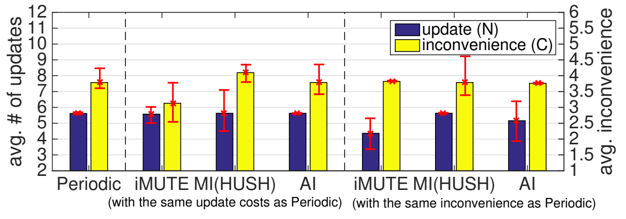
In Fig. 7, we plot the inter-launching time distributions of two chosen participants and show that they are heavy-tailed. We verify by two goodness of fit tests, Cramer-Smirnov-Von-Mises (CSVM) [24] and Akaike [25] that the inter-launching times of all users have the best fit with *Weibull distributions* rather than exponential, log-normal, truncated Pareto, gamma, and Rayleigh distributions. Note that if the shape parameter (β) of a Weibull distribution is less than 1, it is heavy-tailed. When the inter-launching time is heavy-tailed, its corresponding failure rate is decreasing over the elapsed time (which is often called *negative aging* [16]), as shown in the bottom of Fig. 7.

In Fig. 8 (a), we depict the CDFs of average individual inter-launching times of users, where the average inter-launching time is about 143 minutes. We also depict the CDF (cumulative density function) of shape parameters (β) of the fitted Weibull distributions of 26 users in Fig. 8 (b), which indicates that all users show heavy-tailed (i.e., $\beta < 1$) inter-launching times.

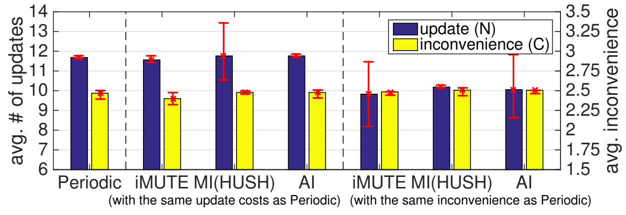
B. Compared algorithms

Periodic policy: This policy updates contents periodically.

Multiplicative increase (MI) policy (HUSH [8]): The MI policy have two parameters: an initial period T_1 and in-



(a) For the cost function $c(a) = \frac{10}{1+\exp(\frac{300-a}{30})}$.



(b) For the cost function $c(a) = a^{0.2}$.

Fig. 9: Expected update and inconvenience costs of *iMUTE*, periodic, MI (HUSH), and AI policies, for two types of inconvenience costs: (a) logistic, and (b) iso-elastic functions. The error bars indicate 25th and 75th percentiles.

creasing parameter σ, where its update interval increases as $T_k - T_{k-1} = T_1 \cdot \sigma^{k-1}$ for $k \geq 2$. MI is used in HUSH [8].

Additive increase (AI) policy: The AI policy is defined by two parameters, an initial period T_1 and increasing parameter δ. Then, the additive policy is $T_k - T_{k-1} = T_1 + (k-1) \cdot \delta$ for $k \geq 2$. For fair comparison, we choose the best parameters of MI/AI policies for a given constraint.

C. Results

For each user, we use the inter-launching time data during the first week to train the *iMUTE* policy for each individual. The remaining inter-launching time data during the second week are exploited as the testing dataset. In particular, we first use the best fitting Weibull distribution of training dataset to represent the inter-launching time distribution of each user. Then, we calculate the failure rate function $r(\cdot)$ according to the fitting Weibull distribution. Given the failure rate function as input, we apply our *iMUTE* algorithm to find the *iMUTE* policy under the cost function $c(\cdot)$. Then, we apply the *iMUTE* policy, periodic policy, and MI/AI policies respectively to make update decisions for each inter-launching interval in the testing dataset. In all cases, the numbers of iterations to converge are less than 10 iterations, for a tolerance parameter $\epsilon = 0.1$. The performance of the policies is measured by average update cost and average user inconvenience.

In Fig. 9, we compare our *iMUTE* policy with periodic policies and MI/AI policies with the same update cost and same user inconvenience. To make the comparison fair, we tune the initial period and increasing parameters of MI/AI policies to achieve the best performance. We compare the performance results under two types of user inconvenience functions: (1) logistic function[11] $c(a) = \frac{10}{1+\exp(\frac{300-a}{30})}$ and (2) iso-elastic function $c(a) = a^{0.2}$. For many mobile applications, contents of applications (e.g., stock price/weather information, coupons) may be volatile and become almost worthless after some time. We use a logistic function that can capture the user inconvenience of such applications. We also consider an iso-elastic function whose exponent is smaller than 1 to model applications and users that have decreasing marginal inconvenience as age increases (i.e., concave inconvenience cost function). The iso-elastic function is widely used in economic analysis to model user behaviors, where it has a constant elasticity.[12]

Under the logistic function case, *iMUTE* decreases the update cost by 28.7%, 29.1%, and 18.2% compared to the periodic, MI, and AI policies respectively, for the same average user inconvenience. The parameters used for the MI policy are $T_1 = 60$ and $\sigma = 5$, and the parameters for the AI policy are $T_1 = 60$ and $\delta = 2500$. For the same average update costs, our policy decreases the user inconvenience by 20.9%, 30.8%, and 20.9%, compared to the periodic, MI, and AI policies, respectively. The parameters for the MI policy are $T_1 = 60$ and $\sigma = 5$, and the parameters for the AI policy are $T_1 = 60$ and $\delta = 700$. In all cases, our policy outperforms the others with lower user inconvenience and update costs.

Under the iso-elastic function case, *iMUTE* decreases the update cost by 18.9%, 3.7%, and 2.3%, compared to the periodic, MI and AI policies respectively, with the same average user inconvenience. The parameters used for the MI policy are $T_1 = 300$ and $\sigma = 1.015$, and the parameters for the AI policy are $T_1 = 300$ and $\delta = 10$. On the other hand, when the same average cost is pursued, our policy decreases the user inconvenience by 2.9%, 3.3%, and 3.3%, compared to the periodic, MI and AI policies, respectively. For this setting, the parameters of the MI policy are set to be $T_1 = 300$ and $\sigma = 1.02$, and the parameters for the AI policy are $T_1 = 300$ and $\delta = 12$. Since the marginal increase shrinks quickly over time in the iso-elastic function, the user inconvenience gain is relatively small, but the reduction in the update cost is still significant compared to the periodic policy.

VII. Discussion

Throughout the paper, we focused on finding an optimal update policy for given user statistics. For the actual system implementation on mobile devices, there are several issues to be discussed such as *trace logging and processing*, *policy re-computation*, and *policy execution*.

First, the inter-launching times are logged for each launching event. After sufficient traces are collected, a distribution is fitted with the best one (Weibull in our case). Then, the optimal update policy is obtained from our *iMUTE* algorithm for the given distribution of the inter-launching time X. In case the underlying distribution changes over time from the change of user behaviors, we may need to use sliding window or discounted weighting such as in [26] to refresh the distribution

[11]The logistic function gives a common S-shaped curve. The chosen parameters show steep increase around 300.

[12]The elasticity of a function $c(x)$ is defined as $a\frac{c'(a)}{c(a)}$. Intuitively, it is the ratio of the relative (percentage) change in the function value $c(x)$ with respect to the relative change in its input x.

and re-compute the optimal update policy. This may happen rarely in practice (e.g., once a week) since people tend to stick to a certain pattern of usage after they get familiarized with the device and the application [27], and therefore the computational overhead of the device becomes more negligible in the long run. Also, in order to prevent *cold start* from the learning period, one may want to use collaborative filtering to obtain statistics from other users with similar attributes, which is out of the scope of this paper.

For the energy-efficient execution of the optimal update policy, we can exploit timer APIs (Application Program Interface) of mobile devices, which wakes a device up at a desired moment with little energy cost. This timer mechanism is efficiently implemented in most mobile OSs such as Android and iOS, and thus incurs minor energy overhead.

VIII. Conclusion

In this paper, we develop an optimal update policy for the applications that prefetch contents in the background with a pull-based manner, under an energy constraint. We model the problem as a constrained MDP (**C-MDP**), and design a two-step solution. We first develop a threshold-based backward induction algorithm to solve an unconstrained MDP (**U-MDP**), and then, we develop an iterative algorithm, *iMUTE* based on the Dekker's method, which converges superlinearly.

We consider two extensions for future work. The first is to extend our framework to cover multiple applications, in which we can further reduce energy by bundling multiple updates from different applications. Finding an optimal policy can be much more complex as the usage patterns can be diverse across multiple applications. The second is to exploit surrounding informations so-called contexts such as the time of a day or the previously used application. A few recent measurement studies [10], [28]–[30] have revealed that the correlations between usage patterns and the surrounding information can be used to accurately predict the inter-launching time X.

We remark that the proposed algorithms in this paper are easily extendable to the resource-constrained inspection and maintenance problems with small modifications. Potential applications include inspection/replacement schedules for aging parts of a machine (e.g., car tires), and clock sync updates in battery-limited sensors with clock drift.

References

[1] S. Rosen, A. Nikravesh, Y. Guo, Z. M. Mao, F. Qian, and S. Sen, "Revisiting network energy efficiency of mobile apps: Performance in the wild," in *Proc. of ACM Internet Measurement Conference*, 2015.

[2] X. Chen, N. Ding, A. Jindal, Y. C. Hu, M. Gupta, and R. Vannithamby, "Smartphone energy drain in the wild: Analysis and implications," in *Proc. of ACM Sigmetrics*, 2015.

[3] Avast, "Avast android app performance & trend report for q3 2016," 2016, http://files.avast.com/files/marketing/materials/androidappperformancereportq32016.pdf.

[4] 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 *Proc. of ACM MobiSys*, 2012.

[5] 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 *Proc. of ACM EuroSys*, 2012, pp. 29–42.

[6] Monsoon Power Monitor, 2016, https://www.msoon.com/LabEquipment/PowerMonitor.

[7] Android Developers, "Optimizing for Doze and App Standby," 2016, http://developer.android.com/intl/ko/training/monitoring-device-state/doze-standby.html.

[8] X. Chen, A. Jindal, N. Ding, Y. C. Hu, M. Gupta, and R. Vannithamby, "Smartphone background activities in the wild: Origin, energy drain, and optimization," in *Proc. of ACM MobiCom*, 2015.

[9] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," in *Proc. of ACM MobiSys*, 2010.

[10] J. Lee, K. Lee, E. Jeong, J. Jo, and N. B. Shroff, "Context-aware application scheduling in mobile systems: what will users do and not do next?" in *Proc. of ACM UbiComp*, 2016, pp. 1235–1246.

[11] A. Mehrotra, V. Pejovic, J. Vermeulen, R. Hendley, and M. Musolesi, "My phone and me: Understanding people's receptivity to mobile notifications," in *Proc. of ACM CHI*, 2016.

[12] T. Dekker, "Finding a zero by means of successive linear interpolation," *Constructive aspects of the fundamental theorem of algebra*, pp. 37–51, 1969.

[13] R. E. Barlow, L. C. Hunter, and F. Proschan, "Optimum checking procedures," *Journal of the society for industrial and applied mathematics*, vol. 11, no. 4, pp. 1078–1095, 1963.

[14] A. Munford and A. Shahani, "A nearly optimal inspection policy," *Journal of the Operational Research Society*, vol. 23, no. 3, pp. 373–379, 1972.

[15] P. L'Ecuyer and J. Malenfant, "Computing optimal checkpointing strategies for rollback and recovery systems," *IEEE Transactions on computers*, vol. 37, no. 4, pp. 491–496, 1988.

[16] J. Jeong, Y. Yi, J.-w. Cho, D. Y. Eun, and S. Chong, "Wi-fi sensing: Should mobiles sleep longer as they age?" in *Proc. of IEEE INFOCOM*, 2013.

[17] A. Grall, C. Bérenguer, and L. Dieulle, "A condition-based maintenance policy for stochastically deteriorating systems," *Reliability Engineering & System Safety*, vol. 76, no. 2, pp. 167–180, 2002.

[18] R. E. Barlow and F. Proschan, *Mathematical theory of reliability*. SIAM, 1996.

[19] E. Altman, *Constrained Markov decision processes*. CRC Press, 1999, vol. 7.

[20] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol. II*. Athena Scientific, 2007.

[21] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley & Sons, 2007, vol. 703.

[22] C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations," *Mathematics of computation*, vol. 19, no. 92, pp. 577–593, 1965.

[23] I. K. Argyros and S. Hilout, *Computational methods in nonlinear analysis: efficient algorithms, fixed point theory and applications*. World Scientific, 2013.

[24] W. T. Eadie, M. G. Roos, and F. E. James, *Statistical Methods in Experimental Physics*. Elsevier Science and Technology, 1971.

[25] K. P. Burnham and D. R. Anderson, "Multimodel inference: Understanding aic and bic in model selection," *Sociological Methods and Research*, vol. 33, no. 2, pp. 261–304, 2004.

[26] A. Garivier and E. Moulines, "On upper-confidence bound policies for switching bandit problems," in *Proc. of Algorithmic Learning Theory*, 2011.

[27] A. Rahmati and L. Zhong, "Studying smartphone usage: Lessons from a four-month field study," *IEEE Transactions on Mobile Computing*, vol. 12, no. 7, pp. 1417–1427, 2013.

[28] C. Shin, J.-H. Hong, and A. K. Dey, "Understanding and prediction of mobile application usage for smart phones," in *Proc. of ACM UbiComp*, 2012.

[29] A. Rahmati, C. Shepard, C. Tossell, L. Zhong, and P. Kortum, "Practical context awareness: Measuring and utilizing the context dependency of mobile usage," *IEEE Transactions on Mobile Computing*, 2014.

[30] J. Lee, K. Lee, E. Jeong, J. Jo, and N. B. Shroff, "Cas: Context-aware background application scheduling in interactive mobile systems," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1013–1029, 2017.

Appendix A
Proof of Lemma 4.1

We consider a continuous-time problem of **P1**, where **P1** is equivalent to the continuous-time problem as the time slot becomes infinitely small. In the continous-time version, $\vec{T} = (T_1, T_2, \cdots)$ are continous variables. We let $T_0 = 0$. Here, we abuse some notations in out discrete-time model for the ease of explanation. We re-define $f_X(t)$ and $F_X(t)$ as the probability density function and cumulative distribution function of the inter-launching time X. Similar to the MDP model, we let $T_{\max}$ be the maximum time duration that the system tracks, and $r_X(t) = \frac{f_X(t)}{\bar{F}_X(t)}$ be constant for $t \geq T_{\max}$. In other words, $T_{\max}$ is $t_{\max}$ multiplied by the length of a time slot. We remove the subscript X for simplicity, in the rest of the proof. $N(\vec{T})$ and $C(\vec{T})$ are the expected update and inconvenience costs as follows:

$$N(\vec{T}) = \sum_{n=1}^{\infty} \bar{F}(T_n),\ C(\vec{T}) = \sum_{n=0}^{\infty} \int_0^{T_{n+1}-T_n} c(t) f(t+T_n) dt,$$

where $\bar{F}(t) = 1 - F(t)$ is the complementary cumulative distribution function. The Lagrangian relaxed continuous-time problem **P2** (which corresponds to **U-MDP**) is as follows:

$$\textbf{P2:} \min_{\vec{T}} g(\vec{T}) = \min_{\vec{T}} C(\vec{T}) + \lambda \cdot N(\vec{T}), \tag{5}$$

where λ is a Lagrange multiplier.

We first derive the first-order necessary condition for optimal update points, $\vec{T}^*(\lambda) = (T_1^*(\lambda), T_2^*(\lambda), \cdots)$ for given λ. For notational simplicity, we use $T_n = T_n^*(\lambda)$ for optimal update points, in the rest of the proof. From the first-order condition (i.e., $\frac{\partial g(\vec{T})}{\partial T_n} = 0$),

$$f(T_n)\Big(c(T_n - T_{n-1}) - \lambda - \int_{T_n}^{T_{n+1}} \frac{f(t)}{f(T_n)} c'(t - T_n) dt\Big) = 0,$$

for $n \geq 1$, where $c'(t)$ is the derivative of the user inconvenience function $c(t)$. Dividing by $f(T_n)$ yields

$$\lambda = c(T_n - T_{n-1}) - \int_0^{T_{n+1}-T_n} \frac{f(t+T_n)}{f(T_n)} c'(t) dt. \tag{6}$$

Then, we differntiate both sides by λ. Note that optimal $\vec{T}$ is dependent on λ. We have

$$1 = \frac{\partial T_n}{\partial \lambda} \frac{a_n}{f(T_n)} - \frac{\partial T_{n-1}}{\partial \lambda} \frac{b_{n-1}}{f(T_n)} - \frac{\partial T_{n+1}}{\partial \lambda} \frac{b_n}{f(T_n)}, \tag{7}$$

where

$$a_n = f(T_n)\Big(c'(T_n - T_{n-1}) + \frac{f(T_{n+1})}{f(T_n)} c'(T_{n+1} - T_n) - \int_0^{T_{n+1}-T_n} \frac{\partial}{\partial T_n}\Big(\frac{f(t+T_n)}{f(T_n)}\Big) c'(t) dt\Big),$$

$$b_n = f(T_{n+1}) c'(T_{n+1} - T_n),\ \text{for } n \geq 1,\ \ b_0 = 0,$$

Now, we derive a series expression for the derivative of $N^o(\lambda) = N(\vec{T}^*(\lambda))$. From the definition of $N(\vec{T})$,

$$-\frac{\partial N^o(\lambda)}{\partial \lambda} = f(T_1)\frac{\partial T_1}{\partial \lambda} + f(T_2)\frac{\partial T_2}{\partial \lambda} + \cdots. \tag{8}$$

By applying the recursive formula in Eq. (7),

$$-\frac{\partial N^o(\lambda)}{\partial \lambda} = e_1 + e_2 + \cdots,\ \text{for } e_n = \frac{d_n^2}{c_n}, \tag{9}$$

$$d_n = f(T_n) + \frac{b_{n-1}}{c_{n-1}} d_{n-1},\ c_n = a_n - \frac{b_{n-1}^2}{c_{n-1}},\ \text{for } n \geq 1,$$

and $d_0 = 0, c_0 = 1$. In the following lemma, we prove that the series in Eq. (9) converges by the ratio test. Therefore, the derivative $\frac{\partial N^o(\lambda)}{\partial \lambda}$ is bounded, which concludes the proof.

Lemma A.1: $\lim_{n\to\infty} \left|\frac{e_{n+1}}{e_n}\right| < 1$, and the series $e_1 + e_2 + \cdots$ converges absolutely.

Proof: From the Lemma B.2,

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n} = \lim_{n\to\infty} \frac{b_n^2}{c_n c_{n+1}} = \lim_{n\to\infty} \frac{b_n}{b_{n-1}} < 1.$$

■

Appendix B
Proof of Lemmas B.1 and B.2

From the first-order necessary condition in Eq. (6), for any n, $T_n - T_{n-1} \geq \tau$ for $\tau = \inf\{\tau | c(\tau) = \lambda\}$ since the integral term is positive. Therefore, there exists sufficiently large n such that $T_m > T_{\max}$ for any $m \geq n$. For the update points after $T_{\max}$, we prove the following lemma.

Lemma B.1: For any n, m such that $T_m > T_{\max}$ and $T_n > T_{\max}$, (i) $\frac{\partial}{\partial T_n}\Big(\frac{f(t+T_n)}{f(T_n)}\Big) = 0$, (ii) $a_n = b_n + b_{n-1}$, (iii) $T_{m+1} - T_m = T_{n+1} - T_n$, and (iv) $\frac{b_{m+1}}{b_m} = \frac{b_{n+1}}{b_n} < 1$.

Proof: (i) For $t \geq T_{\max}$, $r(t) = \frac{f(t)}{1-F(t)} = r$. Thus, $f(t) = r(1 - F(t))$. By differentiating both sides, $f'(t) = -r \cdot f(t)$, i.e., $\frac{f'(t)}{f(t)} = -r$. The partial derivative of $\frac{f(t+T_n)}{f(T_n)}$ over T_n is

$$\frac{\partial}{\partial T_n}\Big(\frac{f(t+T_n)}{f(T_n)}\Big) = \Big(\frac{f'(t+T_n)}{f(t+T_n)} - \frac{f'(T_n)}{f(T_n)}\Big)\frac{f(t+T_n)}{f(T_n)} = 0.$$

(ii) From (i), $a_n = b_n + b_{n-1}$.

(iii) Since $f'(t) = -rf(t)$ for $t \geq T_{\max}$, $f(t) = C_1 \cdot e^{-rt}$ and $1 - F(t) = \frac{C_1}{r} e^{-rt}$ for some constant C_1. Then, **P2** can be separated into two parts before T_n and after T_n:

$$\min_{\vec{T}} g(\vec{T}) = \sum_{l=1}^{n} \left\{\int_{T_{l-1}}^{T_l} c(t - T_{l-1}) f(t) dt + \lambda \cdot (1 - F(T_l))\right\} + e^{-rT_n}\Big(\int_0^{T_{n+1}-T_n} c(t) C_1 e^{-rt} dt + \lambda \cdot \frac{C_1}{r} e^{-r(T_{n+1}-T_n)}\Big) + \cdots.$$

Since the latter part is the same for n or m, the optimal first interval in the latter part (i.e., $T_{n+1} - T_n$ and $T_{m+1} - T_m$) is the same.

(iv) Recall that for $t \geq T_{\max}$, $f(t) = C_1 e^{-rt}$ Then, from (iii), $\frac{b_{n+1}}{b_n} = \frac{f(T_{n+2})c'(T_{n+2}-T_{n+1})}{f(T_{n+1})c'(T_{n+1}-T_n)} = \frac{f(T_{n+2})}{f(T_{n+1})} = e^{-r(T_{n+2}-T_{n+1})} < 1$. Also, from (iii), $\frac{b_{m+1}}{b_m} = \frac{b_{n+1}}{b_n} < 1$. ■

Lemma B.2: $\lim_{n\to\infty} \frac{c_n}{b_n} = \frac{b_{n-1}}{b_n}$ and $\lim_{n\to\infty} \frac{f(T_{n+1})}{d_n} = 0$.

Proof: Consider sufficiently large n such that $T_n > T_{\max}$. From $a_n = b_n + b_{n-1}$, $\frac{c_n}{b_n} = 1 + C_2 - \frac{C_2}{\frac{c_{n-1}}{b_{n-1}}}$, where $C_2 = \frac{b_{n-1}}{b_n} > 1$ (from (iv) of Lemma B.1). By solving the above recurrence equation for $\frac{c_n}{b_n}$, we have $\frac{c_n}{b_n} = 1 + \frac{(-1)^n (C_2-1) C_2 C_3}{\left(-\frac{1}{C_2}\right)^n + (-1)^n C_2 C_3}$, for some constant C_3. Since $C_2 > 1$, $\lim_{n\to\infty} \frac{c_n}{b_n} = 1 + (C_2 - 1) = \frac{b_{n-1}}{b_n}$. Using the above equation,

$$\lim_{m\to\infty} \frac{f(T_{m+1})}{d_m} = \lim_{m\to\infty} \frac{f(T_{m+1})}{f(T_m) + \frac{b_{m-1}}{b_{m-2}} f(T_{m-1}) + \cdots} = \lim_{m\to\infty} \frac{f(T_{m+1})}{(m-n) f(T_m) + f(T_n) + \cdots} = 0.$$

■