Des Autom Embed Syst (2010) 14: 131–159
DOI 10.1007/s10617-010-9054-2

Markov decision process (MDP) framework for software power optimization using call profiles on mobile phones

Eric Jung · Frank Maker · Tang Lung Cheung · Xin Liu · Venkatesh Akella

Received: 22 January 2010 / Accepted: 12 May 2010 / Published online: 25 June 2010

Abstract We present an optimization framework for delay-tolerant data applications on mobile phones based on the Markov decision process (MDP). This process maximizes an application specific reward or utility metric, specified by the user, while still meeting a talk-time constraint, under limited resources such as battery life. This approach is novel for two reasons. First, it is user profile driven, which means that the user's history is an input to help predict and reserve resources for future talk-time. It is also dynamic: an application will adapt its behavior to current phone conditions such as battery level or time before the next recharge period. We propose efficient techniques to solve the optimization problem based on dynamic programming and illustrate how it can be used to optimize realistic applications. We also present a heuristic based on the MDP framework that performs well and is highly scalable for multiple applications. This approach is demonstrated using two applications: Email and Twitter synchronization with different priorities. We present experimental results based on Google's Android platform running on an Android Develepor Phone 1 (HTC Dream) mobile phone.

Keywords User-profile driven · Power optimization · Talk time extension · Markov-decision process · Android · Mobile phones

E. Jung · F. Maker (✉) · V. Akella
Dept. of Electrical and Computer Engineering, University of California, Davis, CA 95616, USA
e-mail: flmaker@ucdavis.edu

E. Jung
e-mail: eajung@ucdavis.edu

V. Akella
e-mail: akella@ucdavis.edu

T.L. Cheung · X. Liu
Dept. of Computer Science, University of California, Davis, CA 95616, USA

T.L. Cheung
e-mail: ctlcheung@ucdavis.edu

X. Liu
e-mail: liu@ucdavis.edu

1 Introduction

Mobile phones are ubiquitous embedded systems. With every generation their user base and computing capability is growing. On September 28, 2009, Apple reported that over three billion iPhone applications were downloaded in just over a year [5]. Recently, Google introduced the Nexus One Android smartphone which has a 1 GHz processor and 512 MB of RAM [11]; Blackberry, Nokia and others are responding in kind with their own application stores and competitive hardware. The larger narrative of these developments recalls the 80's and 90's when we saw the rise of the desktop personal computer, whereas now we are entering the age of the smartphone which will last for the next decade and beyond. Consequently, there is a growing demand for software design and optimization techniques to support this burgeoning platform. Many of the most exciting applications for these devices combine location and web services, and require regular communication with internet services.

A mobile phone has key differences when compared to a desktop or a notebook computer. First, a mobile phone is an *embedded system* as opposed to a general-purpose computer; its primary function (to most users) is voice communication. Secondly, mobile phones are more size and weight-constrained than PCs (including notebooks, netbooks, etc.) and consequently have more limited computing resources than their larger PC counterparts. Therefore, applications have to be more conscious of their resource consumption, especially with respect to main memory usage and battery life. Also, mobile phone systems are event-driven or *reactive*, which means the same application is often launched many times and may only be active for a short duration (e.g. minutes instead of hours or days). Cellphones also typically have one user, unlike a PC which might be shared. Lastly, cell phones are often carried at all times with the user, meaning that it is a far more personal device than even highly mobile laptops. All these differences mean that there is a great opportunity to *customize* the application to an individual's usage patterns, which could be very diverse from one user to another. Users may have different preferences of when they charge their cellphones, their access and availability to Wi-Fi and cellular networks, and usage patterns (for example: heavy call load, internet usage or SMS messages), etc. So, a *one-size-fits-all* approach to application development, which is prevalent in the personal computer environment, is not necessarily appropriate for developing embedded software on mobile phones.

We present a technique for developing and *optimizing* applications to address these concerns. This approach is novel for two reasons. First, it is user profile driven, which means that the user's history is an input to help predict and reserve resources for future talk-time. It is also dynamic: an application will adapt its behavior to current phone conditions such as battery level or time before the next recharge period. Specifically, we propose a Markov Decision Process (MDP) based framework for dynamic optimization of applications running on a mobile phone. This framework allows applications to incorporate user preference and user profiles into *decision making* at run time. Decisions are then made about when resources are used in order to optimize a developer-specified utility or reward function.

In this paper we focus on the energy consumption of an individual application, which in turn affects the battery life and hence the overall talk-time of the mobile phone. We argue that this energy consumption is critically important because (as noted above) the primary purpose of a mobile phone is to provide voice communication. Therefore, it is important that *other* applications such as email clients and browsers do not consume too much energy, thereby undermining the primary function of the phone. This is a key difference between optimizing energy for applications running on mobile phones versus more general software energy optimization. This is explicitly taken into account by our problem formulation, where

the desired talk time is used as an input parameter. We model this by setting a discharge time T that we expect the battery to last, with the underlying assumption that the phone will be recharged after that time. The optimizer attempts to maximize the utility (reward) of different applications while assuring that the battery lasts until at least time T.

There is a subtle but important difference between the proposed work and the traditional problem of optimizing energy on embedded systems, which maximizes battery life by reducing performance via dynamic voltage and frequency scaling to meet a real-time constraint. For example in [14, 15, 19, 20] researchers present a cross-layer optimization methodology for video decoding by dynamically scaling the voltage and frequency of the underlying processor, such that the task of decoding a given video frame finishes *just in time* to meet the time requirement. This time constraint can be predicted based on the time required to process a frame derived on the previous history. Although there is a time constraint in both cases, in our work it is a macro-level (i.e. global) constraint that applies to all tasks and applications that run between now and the user specified time T. Parameter T, which is assumed to be given in this paper, is user-profile driven and can be estimated with good accuracy as shown in [17].

In summary, the key contributions of this paper are:

1. A methodology for *dynamic* power optimization of applications to prolong the battery life time of a mobile phone until a user specified time while maximizing a user defined reward function.
2. A mathematical formulation of the problem via Markov decision processes (MDPs) and techniques to reduce the size of the decision tables.
3. A formulation that optimizes over multiple applications, including a scalable heuristic.
4. An implementation of this technique in two applications based on the Android software development platform.

The rest of the paper is organized as follows. In Sect. 2 we present the problem formulation. First, we introduce the general concept of MDP, develop mathematical formulations using two case studies, and present techniques to solve the mathematical formulations efficiently. We also introduce the concept of an *energy threshold*, and present a heuristic based on this threshold that performs similarly to the MDP framework, and is highly scalable for multiple applications. In Sect. 3 we present the experimental setup employed and show how key parameters are estimated directly from the mobile phone hardware running the Android mobile OS. In Sect. 4 we present results from two case studies that use the proposed optimization framework, as well as results using the heuristic policy. These sections also describe how the user profile data is generated and used within the framework and how the optimization procedure is performed in conjunction with other applications. In Sect. 5 we review related work from the literature, and finally in Sect. 6 we summarize the key ideas of the paper.

2 Problem formulation

We first present the general framework of MDP and then present two case studies. In both cases, we consider voice communication as a high priority service and other delay-tolerant data applications as low priority services. Other high priority applications, particularly user-initiated ones such as browsing or gaming, could also be included relatively easily, but are left out here for simplicity. Our objective is to minimize the disruption to voice communication due to energy depletion from other lower priority applications. When the battery has

sufficient remaining energy with respect to the expected time until the next recharging cycle, other applications can run with higher quality and/or less delay, which consumes more battery. On the other hand, if the remaining battery is low with respect to expected charging time, the system should conserve energy for voice communication at the expense of lower priority services. All the information related to charging time, voice communication patterns, etc. are user profile driven.

We study two generic tasks that are useful to a wide variety of applications on mobile phones: Email data synchronization and combined Twitter and Email synchronization. Data synchronization ensures that local content (i.e. data) is consistent with content from a remote server, such as email messages or Twitter feeds.

2.1 Markov decision process

The Markov decision process (MDP) is a widely used mathematical framework for modeling decision-making in situations where the outcomes are partly random and partly controllable. A MDP is a discrete time stochastic control process represented by a tuple of four objects (S, A, P_a, R_a).

S is the state space, where $s \in S$ is the current state, known to users. In this paper, the state includes the current time, remaining battery energy, time since last synchronization, etc.

A is the action space, where $a \in A$ is the action taken based on the current state. For example, in this paper, the action could be whether to synchronize email or remain idle.

$P_a(s)$ is the probability that action a in state s at time t will lead to state s' at time $t + 1$. Note that this transition is partly random (e.g. due to the random arrival of voice calls) and partly under control since it is based on action a.

R_a is the immediate reward of action a. For example, if the action is to synchronize email, we receive an immediate reward. If the action is to not synchronize email, then the immediate reward is zero.

Our objective in this paper is to maximize the cumulative reward until the expected battery charging time. We note that these rewards are meant to achieve a desired performance on the phone, and are therefore tunable to achieve desired behaviors. For example, if calls are less valuable than data synchronization, the reward for calls can be lowered with respect to data reward. We propose to use the MDP approach to better handle the dynamics of the system because phone calls are stochastic.

In this formulation, an optimal action depends on the current state, the immediate reward and the future reward. For instance, the decision of whether to synchronize email depends on the current state (time, remaining battery and the time since last synchronization). The decision to synchronize email will yield an immediate reward, at the cost of energy consumption which may reduce a future reward. All these factors must be considered in this decision problem.

The main challenge of MDP modeling is to manage its complexity in terms of the number of states, the number of actions and the time horizon. This is important because ultimately the optimal decision procedure, typically in the form of a precomputed decision table, will itself be running on a resource-constrained device (i.e. a mobile phone). The MDP model will not be feasible if it requires too much memory, computation time or energy.

There is typically a tradeoff between the number of states (i.e. granularity) and the computational complexity. We show that in some cases the structure of the problem can be exploited to represent the optimal policy with a reduced state space, e.g. using a threshold-based format.

In our case, the number of actions is limited and the time horizon is finite. Time is slotted and each slot is a time unit. At the beginning of the time slot a decision is made based on the current state information. For example, in our email synchronization application, each time slot is set to one minute. We can also use a coarser granularity of time to reduce the state space.

2.2 MDP model for data synchronization

We consider data synchronization applications that are delay tolerant (e.g. email, calendar, contacts, Facebook pages, RSS feeds, etc.). If the phone is close to its expected charging time and has abundant energy left, we can perform data synchronization more often. However, if the charging time is too far in the future, we should conserve energy by reducing the frequency of data synchronization. Using email as an example, we study how to control the synchronization frequency to maximize user experience. Our objective is to synchronize email as often as possible, while conserving sufficient energy for voice communication. We employ the following notations:

t	current time
T	phone recharge time
E_r	remaining energy at the current time
τ	time elapsed since last synchronization
e^c	energy consumption per unit time of voice call
L^c	length of a voice call, a random variable
R^c	reward of one unit of voice call
$p^c(t)$	Pr[voice call arrives in a time unit]
$R^s(\tau)$	reward of mail synchronization, subadditive
e^s	energy consumed for data synchronization
E_{L^c}	expectation over L^c
$f_r(E_r)$	reward for the remaining energy at charging time T

In addition, $\mathbf{1}\{\cdot\}$ is an indicator function,

$$\mathbf{1}\{x\} = \begin{cases} 1 & \text{if } x \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

Last, we denote $x^+ = \max(0, x)$.

We assume that $R^s(\cdot)$ is an increasing subadditive function. It is increasing because the need to synchronize is larger if there has been a longer delay. It is subadditive so that the following property is satisfied:

$$R^s(x) + R^s(y) \geq R^s(x + y), \quad x, y \geq 0.$$

The property indicates the value of timeliness. Synchronizing twice (left-hand side), which brings information in a more timely manner, is more valuable than once (right-hand side) during the same time interval. Examples of such increasing subadditive functions include $\log(1+x)$ and $\sqrt{1+x}$. Note that T, the (re)charging time, is assumed to be fixed and known in this paper (e.g. 10 pm). It could also be dynamic, obtained based on user profiling [17].

For the MDP framework used for the email application, t, E_r and τ are the input of MDP. The action is whether or not to synchronize email. Our objective is to maximize the total utility (out of the available battery). As we discussed earlier, the optimal action depends on the current state, the immediate reward and the future reward. This is captured in the optimality equation discussed next.

2.2.1 Optimality equation

Let $V(t, E_r, \tau)$ be the optimal value at state (t, E_r, τ). In other words, it is the maximal total reward at the current state optimized over all possible actions, taking into account the future reward. We first define the following notations,

$$\begin{aligned} v^c(t, E_r, \tau) &= E_{L^c}\Bigg[V(t + L^c, (E_r - L^c * e^c)^+, \tau + L^c) \\ &\qquad + \min\left(\left\lfloor \frac{E_r}{e^c} \right\rfloor, L^c \right) R^c \Bigg], \\ v^s(t, E_r, \tau) &= V(t + 1, (E_r - e^s)^+, 1) + R^s(\tau)\mathbf{1}\{E_r \geq e^s\}, \\ v^i(t, E_r, \tau) &= V(t + 1, E_r, \tau + 1). \end{aligned}$$

Explanations are in order. First, $v^c(t, E_r, \tau)$ is the value in the case that a phone call occurs, including both immediate reward and future reward. Recall that L^c is a random variable, representing the length of the phone call. The immediate reward of the phone call is $\min(\lfloor E_r/e^c \rfloor, L^c)R^c$, which is proportional to the length of the phone call supported by the remaining battery energy. We assume that when a phone call occurs, no synchronization activity is allowed. The terms $v^s(\cdot)$ and $v^i(\cdot)$ correspond to the case when no phone call happens in this time slot. In this case, the value is $v^s(\cdot)$ if the action is to synchronize mail and the value is $v^i(\cdot)$ if the action is to stay idle. When the action is to synchronize mail, $R^s(\tau)\mathbf{1}\{E_r \geq e^s\}$ is the immediate reward and $V(t + 1, (E_r - e^s)^+, 1)$ is the future reward. When the action is to stay idle, the immediate reward is zero and the future reward is $V(t + 1, E_r, \tau + 1)$.

We have the following optimality equation:

$$\begin{aligned} V(t, E_r, \tau) &= p^c(t)v^c(t, E_r, \tau) \\ &\qquad + (1 - p^c(t)) \max\left\{ v^s(t, E_r, \tau), v^i(t, E_r, \tau) \right\}. \end{aligned} \tag{1}$$

In other words, when no phone call occurs, the power optimizer can decide whether to synchronize mail or to stay idle, depending on which action results in higher return, considering both immediate and future rewards.

Based on the above formulation, we can solve the problem (i.e. find the optimal decision for each state (t, E_r, τ)) using dynamic programming through backward induction.[1] We also have the following boundary conditions,

$$V(T, E_r, \tau) = f_r(E_r), \tag{2}$$

$$V(t, 0, \tau) = 0. \tag{3}$$

The boundary condition defines the optimal decision at time T, the end of the discharge period, and when the phone is out of battery. The first equation sets the value of remaining energy at the charging time. For simplicity, we set $f_r(E_r) = 0$ in this paper, which implies that the energy remaining at the charging time has no value. This may not be the case, especially

[1] Dynamic programming through backward induction is used to calculate the optimal results from the last time slot, then the second last and so on. This approach is used because the optimal action in an earlier slot depends on the action of a later slot.

Table 1 Threshold-based decision table

Time	E_r	Threshold
77	396	11
77	397	11
77	398	11
77	399	11
77	400	11
78	0	NEVER

if the charging time is not a constant. Given the boundary condition at time T, one can then find the optimal action at time $T-1$. This same process can be used for each tuple (t, E_r, τ) if we have the optimal decision (and the corresponding value) at time $t+1, t+2, \ldots, T$. The solution can be represented using a three-dimensional table where there is an optimal decision for each tuple (t, E_r, τ). The size of the decision table is proportional to the number of states, which is the product of the length of the discharging period, the number of different energy levels, and the number of possible elapsed times from the last synchronization. This could be very large, especially if the granularity of t and τ is small. In the following, we will present a special property of this problem that allows a more structured and simplified solution.

Theorem 1 *Define*

$$\tau^*(t, E_r) = \min\left(\tau : v^s(\tau) \geq v^i(\tau)\right).$$

The following policy is optimal:

$$a = \begin{cases} sync & \tau \geq \tau^*(t, E_r), \\ idle & \tau < \tau^*(t, E_r). \end{cases}$$

The proof of the theorem is presented in Appendix A. The structure is useful in reducing the complexity/memory required for the optimal policy. Instead of a three-dimensional table, one can represent the optimal policy using two dimensions, i.e. for each (t, E_r) tuple, only $\tau^*(t, E_r)$ is needed to represent the optimal decision.

In Table 1, we show a sample of the decision table derived using the measurement data, discussed in detail later. We see that at time 77 (minutes) with remaining energy of 396 units, the threshold is 11. In other words, if the email has not been synchronized for 11 units of time or more, it should synchronize in this time unit. Using the threshold-based policy, the size of the decision table is reduced to the product of the length of the discharging period and the number of different energy levels.

2.3 MDP model for multiple delay-tolerant applications

We now consider a scenario where two delay-tolerant applications, email and Twitter, are to be synchronized based on the MDP framework. While the same intuition from the single application case applies, i.e. synchronization rates should adapt based on battery power and time to next recharge, there are two new issues to be considered, service priority and non-additive data service energies.

The former means that an end user may value the timeliness of certain applications more than others and therefore would want them to be refreshed more often. The latter means

that, due to overhead such as wakeup or wireless radio startup energies, the energy cost of synchronizing two services individually may be higher than synchronizing them at the same time. Additionally, the energy costs for synchronizing each service individually may be different. The main consequence of these new issues is that the set of possible actions expands to include synching either application alone or both applications at the same time, each of which may lead to different utilities for the end user.

We present a high-level overview of the MDP formulation here, leaving the detailed derivations to Appendix B. As in the previous section, our objective is to synchronize data services as often as possible while conserving energy for voice communication, with the added condition that one application may be more valuable than the other. We first number the services (1: Email, 2: Twitter) and extend the previous notation to differentiate these services:

τ_i	time since last synchronization of service i
$R_i^s(\tau_i)$	sync reward for service i, subadditive
e_i^s	energy consumed synchronizing service i
e_b^s	energy consumed synchronizing both services

From the notation, we see that the service priority can be defined mathematically by defining their reward functions $R_i^s(\tau_i)$ separately. The energy costs for each action are also defined separately.

The MDP framework also must be extended to consider the time since the last synchronization for each service, as well as the new actions possible. Therefore, a state is defined by the tuple (t, E_r, τ_1, τ_2) and the optimal value at the state is denoted $V(t, E_r, \tau_1, \tau_2)$. We denote the value of synchronizing services 1, 2 and both as $v_1^s(\cdot)$, $v_2^s(\cdot)$ and $v_b^s(\cdot)$ respectively. We assume that synchronizations can only occur if the user is not on a call. The value for receiving call service and for remaining idle in the absence of call service are still denoted $v^c(\cdot)$ and $v^i(\cdot)$, respectively. We leave the derivation of these functions to Appendix B.

Therefore, for the two application scenario we now have the choice of four different actions: stay idle, sync email, sync Twitter, or sync both applications together. The optimality equation can be stated as

$$\begin{aligned} V(t, E_r, \tau_1, \tau_2) = {} & p^c(t) v^c(t, E_r, \tau_1, \tau_2) \\ & + (1 - p^c(t)) \max\big\{ v^i(t, E_r, \tau_1, \tau_2), v_1^s(t, E_r, \tau_1, \tau_2), \\ & \qquad v_2^s(t, E_r, \tau_1, \tau_2), v_b^s(t, E_r, \tau_1, \tau_2) \big\}. \end{aligned} \tag{4}$$

The decision is chosen that maximizes the value function $V(\cdot)$ and can only be made if there is no call occurring. The initial conditions are analogous to the single service case,

$$V(T, E_r, \tau_1, \tau_2) = f_r(E_r), \tag{5}$$

$$V(t, 0, \tau_1, \tau_2) = 0. \tag{6}$$

Using these sets of equations, a similar backward induction step can be used to obtain the optimal values, and therefore actions, of each state.

2.4 Energy threshold heuristic

As the number of data synchronization services grows, the state space for the MDP formulation grows exponentially, because there must be a state variable to represent the time since

last synchronization for each service. Therefore, it is necessary to determine a heuristic policy that is scalable as the number of services grows. With this motivation, we now present a heuristic policy based on an "energy threshold" concept. The policy is simple to implement and performs well when compared to the optimal MDP-based policies.

The basic idea of the energy threshold is to determine the remaining energy level that is required to serve the future call time after t based on the call profile for every time t. This threshold can then be used to inform the decision of whether or not to synchronize a low priority service at a given time. This energy threshold can be extended to include an energy budget for other high priority applications, particularly user-initiated apps such as web-browsing or gaming. We leave them out here for simplicity of exposition.

Intuitively, as the user approaches the recharge period, less and less energy needs to be reserved for future calls. The basic idea of an energy threshold policy is to compare the phone's remaining energy to the energy threshold at any time, and make a decision to synchronize or not based on this information. If the remaining energy exceeds the predetermined energy threshold at some time, then the phone may choose to sync one or more services based on some policy.

2.4.1 Calculating the energy threshold

To determine the energy threshold, we first calculate the cumulative distribution function (cdf) of the call time received after a time t, for every t in the time horizon. We define C_t as a Bernoulli random variable with probability $p^c(t)$ that the user will be on a call at time t. Denoting $C_t \in \{0, 1\}$ to represent when a user is idle or on a call at time t, we can write

$$Pr[C_t = 1] = 1 - Pr[C_t = 0] = p^c(t).$$

Now we define the random variable X_t as the predicted number of call minutes received after time t,

$$X_t = \sum_{i=t}^{T} C_i.$$

The cdf for X_t can be calculated using backward induction. Denoting $F_{X_t}(\cdot)$ as the cdf of X_t, we then set a probability threshold $p_{th} \leq 1$ and determine the time threshold $\Gamma(t)$ as

$$\begin{aligned} \Gamma(t) &= \arg\max_x \ x \\ \text{subject to} \quad & F_{X_t}(x) \leq p_{th}. \end{aligned}$$

Therefore, with probability p_{th}, there will be no more than $\Gamma(t)$ call minutes after time t based on the calculated cdf. As p_{th} increases, the time threshold will become more conservative. The energy threshold $E_{th}(t)$ is defined as the energy required at time t to guarantee with probability p_{th} that all future calls will be served,

$$E_{th}(t) = e^c \cdot \Gamma(t). \tag{7}$$

This process can easily be extended to account for other types of high priority applications, and would be especially useful for user-initiated applications. For example, if sufficient history for browsing application use exists, this process could be performed for those applications specifically, and added to $E_{th}(t)$.

2.4.2 Synchronization policy for multiple applications based on the energy threshold

We now discuss the design of synchronization policies based on the energy threshold concept. In particular, we define an energy threshold based policy for the Twitter/Email application. This policy is designed to meet the requirements of the multiple application scenario, to maximize synchronization frequency while preserving energy for future voice activity and serving different priority services. We note that while this policy is not comprehensive, it is easily generalized for different priority differences and energy costs, and demonstrates the ability of energy threshold heuristic policies to perform well under the desired goals.

We assume that the user may prioritize email over the Twitter application. Therefore, we enforce a policy in which email may be synchronized alone, but Twitter can only sync periodically with the email application. Since email may have higher priority than Twitter, we assume that the user is willing to spend some extra energy to obtain email more frequently. At the same time, we enforce Twitter as a "piggy-backing" service, i.e. it only synchronizes when some other service does, because the energy of synchronizing both is potentially less than synchronizing each service separately (i.e. $e_b^s \leq e_1^s + e_2^s$). This design choice is motivated by our MDP performance results, which show that the lower priority service is only synchronized jointly with the higher priority service.

We first define a synchronization period M as the period of joint synchronization. We also define N_1 as the number of times the email has synchronized in the current discharge period. Therefore, if $N_1 \bmod M = M - 1$, the next synchronization to occur is a joint synchronization of both services. For a current state (t, E_r, τ_1, τ_2), we then calculate $\tau_{th}(t)$, the threshold at which the data synchronizations should occur,

$$\tau_{th}(t) = \begin{cases} \frac{T-t}{(E_r - E_{th}(t))/e_b^s}, & \text{if } N_1 \bmod M = M-1, \\ \frac{T-t}{(E_r - E_{th}(t))/e_1^s}, & \text{otherwise.} \end{cases} \tag{8}$$

Essentially, $\tau_{th}(t)$ represents the synchronization period that could be handled by the current *energy cushion*, the difference between the current remaining energy and the energy threshold. The divisor represents the number of synchronizations that can be handled with the energy cushion, while the dividend is the remaining time to the end of the discharge period. We note that $\tau_{th}(t)$ only depends on the parameters of service 1 (email) since we assume that Twitter is lower priority and only synchronizes jointly with email. The action decision $a(t)$ is then defined as follows:

$$a(t) = \begin{cases} \text{Sync service 1,} & \text{if } \tau_1 \geq \tau_{th}(t),\ N_1 \bmod M \neq M-1,\ E_r > E_{th}(t), \\ \text{Sync both services,} & \text{if } \tau_1 \geq \tau_{th}(t),\ N_1 \bmod M = M-1,\ E_r > E_{th}(t), \\ \text{Idle,} & \text{otherwise.} \end{cases} \tag{9}$$

The decision is dependent on the τ_1 parameter, since email has priority in this case. If the synchronization period $M = 1$, which means that both services have the same priority, the services always synchronize together. Note that the decision is *always* idle if remaining energy is less than the energy threshold at any given time. This means that the heuristic is always trying to protect the high priority application (voice) to within the p_{th} threshold. We also note that this heuristic can easily be extended to more than two applications.

2.5 High priority applications

In the formulations presented, we have used call time as the high priority application that we wish to protect. However, this may be a shifting paradigm as smart phones become more

Fig. 1 Experimental setup

akin to personal computers than voice communication devices. To this end, our model can be extended to include other user-initiated applications as high priority. For example, in the MDP formulations the energy costs as well as the probability of call arrival $p^c(t)$ can be supplemented by energy costs for different types of applications (e.g. web browsing or gaming) as well as the probability of the user initiating those applications. Expected costs for the use of these apps can also be added to the energy threshold equation, thus protecting them from interference from lower priority functions.

The difficulty arises in characterizing these types of applications. Energy costs will vary greatly depending on the type of application, and depending on frequency of use, reliable expectations of application use may be hard to obtain. We leave the characterization and profiling of such activity to future work.

3 Experimental setup

We use the Android Developer Phone 1 (HTC Dream [12]) to obtain the power measurement data needed for this paper. The Android platform [10] was chosen due to its developer-friendly Java development environment and open source operating system running a modified Linux kernel. In other words, the complete source code was available for modification and inspection during our investigation.

The power consumption of the HTC Dream mobile phone was measured during our experiments by using a DC power supply (Agilent E3644A [1]) to supply power to the phone instead of the phone's battery. Measurement communication was achieved by connecting the power supply for the phone to a laptop computer using the IEEE-488A General Purpose Interface Bus (GPIB). Power measurements were then sampled via a custom Python script using the PyVISA package [7] for easy GPIB communication. We also implemented maximum voltage and current levels to avoid damage to the phone during experimentation. All measurements were performed for a user-specified duration of time, with a sampling frequency of 11.76 Hz, the maximum sampling frequency possible using the PyVISA package.

3.1 Power profiling

3.1.1 Android application

In order to determine the constants for the MDP equations, we ran an application called "Tweemail" (Fig. 2). This program is capable of checking Twitter, Email or both, based

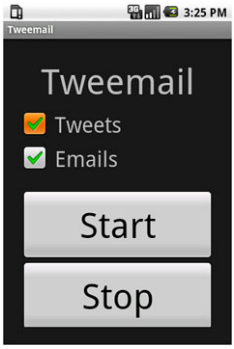

Fig. 2 Tweemail application

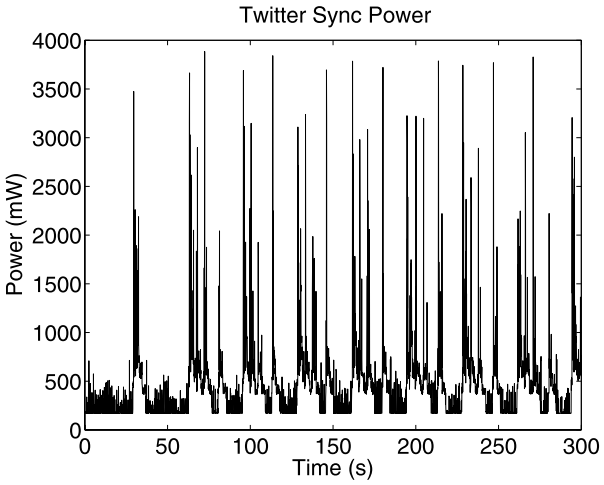

Fig. 3 Twitter power consumption

on user interface preferences. Twitter was synchronized using the Twitter4j library [2] and email was updated using the javamail-android [1] port of the standard JavaMail library to the Android platform. The Twitter client was configured to synchronize with a Twitter feed with subscriptions to the ten most popular Twitter feeds at the time of analysis. The email client synchronized with a Gmail account created for this experiment and fetched only the email envelope information (which most email clients do to conserve bandwidth). A mobile data connection was established using the AT&T EDGE network in the building that houses our laboratories.

Figures 3, 4, 5 show power measurements of Twitter, Email, and joint Twitter and Email synchronization using the "Tweemail" application. All tests were completed with the backlight on during the entire test and no other user initiated processes running. All three combinations of the two synchronization tasks were executed in a loop with 30 second separations between them to facilitate the measurement of each loop's duration. Each power measurement and the time it was measured was recorded and then a discrete integral was performed in order to determine the energy dissipated (Table 2).

These results rank as one would expect, with email requiring slightly more energy than Twitter. This is because of the difference between interfacing with email and Twitter services. Each Twitter message requires only a maximum of 140 characters of data to be downloaded and is performed using a RESTful interface over HTTP. On the other hand, email

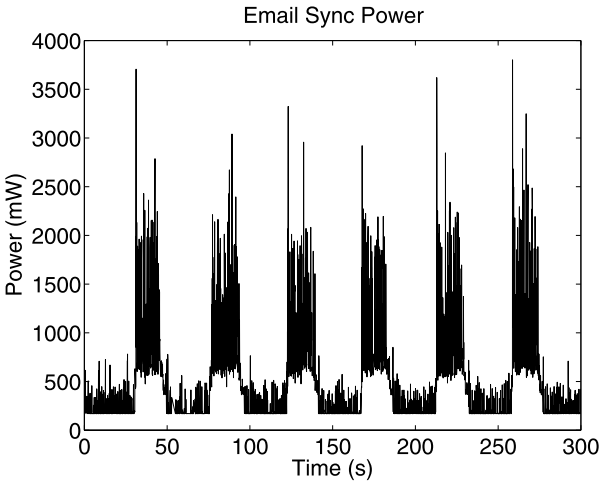

Fig. 4 Email power consumption

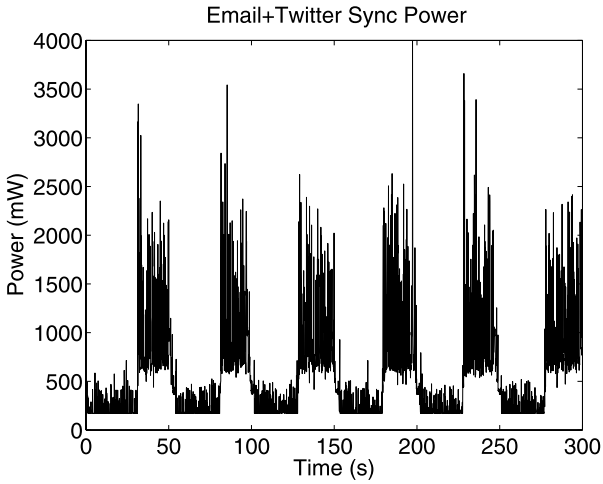

Fig. 5 Twitter and email power consumption

Table 2 Energy for synchronized applications

Application	Energy (J)
Twitter	8.743
Email	11.9889
Twitter + Email	16.9704

employs its own POP3 protocol and each message envelope is significantly longer than a Twitter message (see Appendix C). Therefore, while HTTP and POP3 are relatively similar in terms of their burden on the phone, email requires more energy than Twitter to synchronize due to the length of their messages. Furthermore, due to the inherent overhead incurred in initializing the EDGE radio to communicate with a new server, it is also no surprise that

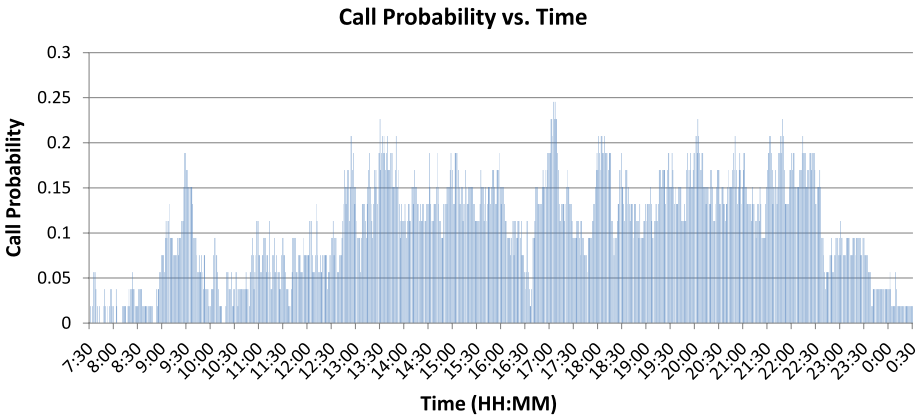

Fig. 6 User voice communication profile

both services updated together consume less energy than the sum total of each operation individually.

3.2 Voice activity profiling

A 66-day call log history was collected.[2] The first 53 days of the history was used to generate the user profile, while the later 13 days were used to run the simulation as 13 different test cases. Figure 6 shows the normalized histogram of the call history of a user from the user's call log. The call log records the time and duration of every phone call in a given period of time while the user profile shows the frequencies of a user making a phone call for each time interval unit within a day. The probability of getting a phone call at a given time, $p^c(t)$, is thus estimated by dividing the number of times a call has occurred at time interval t by the number of days the call log recorded.

4 Results and discussion

We implemented the MDP optimization framework for both the email alone and Twitter/email synchronization applications using the above power measurement data and user profile. The key results are presented next.

4.1 Email synchronization

Experiments were performed to compare the outcomes of using the proposed MDP based email synchronization policy with that of the traditional fixed-frequency email synchronization policy. A piece of the decision table based on the proposed MDP framework using the measurement data is shown in Table 1. For the latter, the phone was simulated to synchronize the email every 5, 10, 20, 30, 40 and 60 minutes. The phone was simulated to discharge from time $t = 0$ to time $t = 960$ (16 hours) and no synchronization would be initiated during a phone call. We assumed that $f_r(E_r) = 0$; i.e. remaining energy at the phone charging time

[2]The usage log can also be generated in realtime and updated periodically.

Table 3 A day with light voice call (20 minutes)

Metric	5	10	20	30	40	60	MDP
N_{syn}	191	95	47	31	23	15	311
Mean	5.02	10.03	20.06	30	40.09	60	3.07
Dev	0.16	0.23	0.32	0	0.42	0	2.82
M_{call}	20	20	20	20	20	20	20
T_{out}	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
$E_r(T)$	131.83	236.46	288.77	306.21	314.93	323.65	1.04

Table 4 A day with moderate voice call (55 minutes)

Metric	5	10	20	30	40	60	MDP
N_{syn}	184	92	47	31	23	15	215
Mean	5.21	10.42	20.19	30.48	40.39	60.20	4.45
Dev	1.52	2.13	0.80	2.34	1.50	0.56	4.88
M_{call}	55	55	55	55	55	55	55
T_{out}	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
$E_r(T)$	34.46	134.73	183.77	201.21	209.93	218.65	0.67

Table 5 A day with heavy voice call (121 minutes)

Metric	5	10	20	30	40	60	MDP
N_{syn}	99	66	43	30	23	15	33
Mean	5.54	11.14	21.05	31.47	41.09	60.53	28.94
Dev	3.25	4.99	4.04	5.07	3.46	2.07	69.38
M_{call}	97	109	117	121	121	121	121
T_{out}	581	746	908	N.A.	N.A.	N.A.	N.A.
$E_r(T)$	0	0	0	4.30	11.93	20.65	1.03

$t = 960$ has no value. Therefore, it is desirable to have the phone use all of its battery life by $t = 959$ (which means that all energy is used for voice call and other services in the day).

In the simulation, we use the following parameters, obtained from the power measurement on an HTC mobile phone running on the Android platform. The initial energy level is 400 units (i.e. $E_r = 400$ at $t = 0$), which can serve around 133 minutes of talk time. Each energy unit is approximately 11 J. Energy consumption for each email synchronization $e^s = 1$. Energy consumption for making a phone call per minute is $e^c = 3$.

We run the simulation in the later 13 days of the user call log (profile shown in Fig. 6) as 13 different test cases. The reward function of email synchronization used was $\sqrt{\tau + 1}$. The entire discharge period was divided into 1-minute time intervals. We present the results in detail for three representative days with light, moderate and heavy voice traffic in Tables 3, 4, 5. To be concise, we omit the results for the other 10 days as they follow the same trend. In the result, N_{syn} is the number of synchronizations performed, *Mean* is the mean of the synchronization period, *Dev* is its standard deviation, M_{call} is the total number of phone call

minutes, T_{out} is the time the battery runs out of energy, and $E_r(T)$ is the remaining energy at the end of the day when the phone recharges.

Under the MDP-based synchronization policy, the phone synchronized more frequently when there were less phone calls, compared to that of the fixed-frequency policy. Among those test cases when the phone did not power off due to insufficient energy before the charging time ($T = 960$), the number of synchronizations made by our policy is always higher or equal to those of any fixed-frequency policy.

Under the MDP policy, the phone only ran out of battery when the actual talk time in the test cases is close to the maximum talk time supported by the battery. In the experiment, it happened in only one day when there was 131 phone-call minutes. In this case, the battery life of the MDP policy is only outlived by that of the 60-minute policy by 1 minute.

In fixed-frequency policy, the standard deviation of the synchronization period is non-zero because no email synchronization occurs when a phone call is taking place. Using our policy, the standard deviation of the synchronization period is much higher than that of fixed-frequency synchronization. The phones running our policy tend to synchronize less frequently near the beginning of the discharging period and tend to synchronize more frequently near the end of the discharging period, especially when the voice usage is light during the day. When the voice usage is (very) heavy, the opposite is observed. To reduce the dispersion of the synchronization frequency, we can set a non-zero reward function for remaining energy at the charging time, or change the reward function to also include the time since the last charging period. This is also reasonable because charging time may vary.

In summary, compared to the fixed frequency synchronization policy, our scheme is dynamic—it allows more synchronization when voice traffic volume is low and reduces data service frequency when voice traffic is heavy. Because of this dynamic nature, it serves the user more effectively by taking into account the priorities of services. There are various directions that we can improve or modify the performance of the proposed MDP scheme. First, to further reduce the chance of missing a phone call (because the phone has depleted its battery before charging time), we can set a non-zero reward function for remaining energy at the charging time; e.g. $f_r(E_r) = c \log(1 + E_r)$ at time T. This value would reward remaining energy at time T and would make data synchronization more conservative.

4.2 Twitter/email synchronization

We now present simulation results for our MDP-based Tweemail synchronization application. Our results show that both the MDP and energy threshold policy in a multiple-application scenario outperforms simple periodic synchronization schemes while protecting user call activity. We also show that MDP and energy threshold policies are viable for a wide range of call arrival rates, and that policies based on real call profiles are robust to wide ranges of call loads.

4.2.1 Real call traces

Similar to the previous section, we obtain the MDP decision table for the two-application scenario based on the real user profile shown in Fig. 6. We then compare the performance of this policy and simple periodic synchronization policies with periods 5, 10, 20, 30 and 60 minutes, where both services are synchronized simultaneously.

The simulation parameters are the same as the previous section, i.e. the discharge time is from $t = 0$ to $t = 960$ and the initial energy level is 400 units, with each unit corresponding to 11 J. We assume that $f_r(E_r) = 0$. Based on the energy measurement results from Sect. 3,

Table 6 A day with light voice call (20 minutes)

Metric	5	10	20	30	60	MDP 1	MDP 2
N_b^s	191	95	47	31	15	220	103
N_1^s	0	0	0	0	0	0	165
$Mean_1$	5.02	10.03	20.06	30.00	60	4.35	3.57
Dev_1	0.16	0.23	0.32	0	0	4.18	3.54
N_2^s	0	0	0	0	0	0	0
$Mean_2$	5.02	10.03	20.06	30	60	4.35	9.29
Dev_2	0.16	0.23	0.32	0	0	4.18	7.14
M_{call}	20	20	20	20	20	20	20
T_{out}	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
$E_r(T)$	45.33	193.44	267.49	292.17	316.86	0.58	1.26

Table 7 A day with moderate voice call (55 minutes)

Metric	5	10	20	30	60	MDP 1	MDP 2
N_b^s	153	92	47	31	15	152	71
N_1^s	0	0	0	0	0	0	114
$Mean_1$	5.23	10.42	20.19	30.48	60.20	6.30	5.17
Dev_1	1.64	2.13	0.80	2.34	0.56	7.22	6.07
N_2^s	0	0	0	0	0	0	0
$Mean_2$	5.23	10.42	20.19	30.48	60.20	6.30	13.48
Dev_2	1.64	2.13	0.80	2.34	0.56	7.22	12.23
M_{call}	54	55	55	55	55	55	55
T_{out}	809	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
$E_r(T)$	0	93.06	162.49	187.17	211.86	0.49	1.21

the energy cost for syncing email and Twitter alone is $e_1^s = 1$ and $e_2^s = 0.8$ respectively, syncing both together is $e_b^s = 1.5$, and the call energy cost remains $e^c = 3$. Note that e_b^s, the cost to synchronize both services at the same time, is less than $e_1^s + e_2^s$, the cost to synchronize both individually, due to the overhead to perform a data synchronization.

For the synchronization reward functions, we use $R_1^s(\tau_1) = C_1\sqrt{\tau_1 + 1}$ and $R_2^s(\tau_2) = C_2\sqrt{\tau_2 + 1}$, where C_1 and C_2 are constants. These constants can be tuned to reflect the different priority levels for different services. For our simulations, we assume that $C_1 + C_2 = 1$ and obtain MDP policies for different C_1/C_2 ratios to prioritize services.

Tables 6, 7, 8 show the performance of two MDP policies and the periodic policies over the same three representative call days presented in the previous section. In the tables, N_1^s/N_2^s are the number of times email/Twitter are synched alone, and N_b^s is the number of simultaneous syncs. $Mean_i/Dev_i$ is the average time and standard deviation for the refresh period of service i. M_{call} is the number of call minutes achieved, T_{out} is the time when the energy is depleted, and $E_r(T)$ is the remaining energy at the end of the discharge period. "MDP 1" corresponds to a policy where $C_1/C_2 = 2$ and "MDP 2" to a policy where $C_1/C_2 = 3$. These ratios are chosen specifically to cause the MDP decision table to treat the two synchronization services with different priorities.

Table 8 A day with heavy voice call (121 minutes)

Metric	5	10	20	30	60	MDP 1	MDP 2
N_b^s	99	59	42	29	15	23	14
N_1^s	0	0	0	0	0	1	14
$Mean_1$	5.54	11.25	21.07	31.52	60.53	39.88	34.18
Dev_1	3.25	5.27	4.08	5.15	2.07	116.48	111.04
N_2^s	0	0	0	0	0	0	0
$Mean_2$	5.54	11.25	21.07	31.52	60.53	41.61	68.36
$Dev(s2)$	3.25	5.27	4.08	5.15	2.07	118.82	163.48
M_{call}	82	102	111	118	121	121	121
T_{out}	566	668	892	933	N.A.	N.A.	N.A.
$E_r(T)$	0	0	0	0	13.86	0.43	0.14

First, we notice that in all periodic cases and the first MDP case, almost all synchronizations of the two services occur simultaneously. This is by design in the periodic cases, but in the "MDP 1" case this reflects the extra energy cost associated with synchronizing each service separately. Although $C_1/C_2 = 2$ in the "MDP 1" case, implying that the user values email more than Twitter, this effect is mitigated by both the lower energy cost of synchronizing Twitter generally, and the lower total energy cost of synchronizing both simultaneously.

However, the "MDP 2" policy ($C_1/C_2 = 3$) shows several instances of email synching alone. This is because for a C_1/C_2 factor this large, reward is maximized by dedicating more energy explicitly to the email service. This effectively prioritizes email service over Twitter. We also note that the mean refresh time for email is shorter for the "MDP 2" case than the "MDP 1" case, which would be expected given its higher priority in the former case. We also notice that for either MDP policy, the Twitter service *never* syncs alone. This again reflects the lower energy cost of synching simultaneously; because Twitter is valued less and synching it together with other services is more cost effective, the MDP policy always chooses this action.

Aside from the Twitter synchronizing behavior, the results here follow the same trends as the email service in Sect. 4.1. We see that in all cases, both MDP policies are able to adapt to the different call volumes better than any of the periodic cases, synching more often when voice activity is light and visa versa when voice activity is heavy. This is reflected in both policies' ability to reach the full call time and deplete the battery to nearly zero at the end of the day. In fact, for all 13 call days, both MDP policies protect voice call at least as well as the conservative 60 minute period policy, while achieving significantly more synchronizations for lighter call days. We also notice that this holds true even if the user prioritizes one service over the other—call time is protected and energy is used effectively.

Therefore, we see that for two applications, the desired goals of the MDP policy are still achieved—call time is protected while synchronization period is adjusted to compliment the call loads. We also show that reward functions can be structured to prioritize one service over another, and the MDP is able to choose which service to synchronize based on this priority and the remaining energy. Further study is needed to determine the effects of non-zero functions for $f_r(E_r)$ and to experiment with different sync reward functions. For example, synchronization reward functions that depend on time t may be able to weight the MDP to synchronize more during certain times, while our current scheme tends to decrease the refresh period as the day goes on.

Table 9 A day with light voice call (20 minutes)

Metric	MDP 1	MDP 2	85%, $M = 1$	85%, $M = 2$
N_b^s	220	103	220	129
N_1^s	0	165	0	129
$Mean_1$	4.35	3.57	4.35	3.71
Dev_1	4.18	3.54	4.66	4.07
N_2^s	0	0	0	0
$Mean_2$	4.35	9.29	4.35	7.42
Dev_2	4.18	7.14	4.66	7.94
M_{call}	20	20	20	20
T_{out}	N.A.	N.A.	N.A.	N.A.
$E_r(T)$	0.58	1.26	0.58	0.38

Table 10 A day with moderate voice call (55 minutes)

Metric	MDP 1	MDP 2	85%, $M = 1$	85%, $M = 2$
N_b^s	152	71	152	89
N_1^s	0	114	0	89
$Mean_1$	6.3	5.17	6.3	5.38
Dev_1	7.22	6.07	7.76	6.71
N_2^s	0	0	0	0
$Mean_2$	6.30	13.48	6.3	10.75
Dev_2	7.22	12.23	7.76	12.93
M_{call}	55	55	55	55
T_{out}	N.A.	N.A.	N.A.	N.A.
$E_r(T)$	0.49	1.21	0.49	0.69

4.2.2 Performance of energy threshold heuristic

We now compare the performance of the energy threshold heuristic policy defined in 2.4.2 to our MDP results. All simulation parameters are the same as in the previous case. In the simulations, we choose a threshold of $p_{th} = .85$ for the energy threshold. This threshold is chosen because it is highly protective of call time while not being overly conservative, and was shown to perform well for this particular user. This parameter could be tuned to suit different types of user profiles. We also simulate over different M, the periods for joint synchronizations.

In Tables 9, 10, 11, we compare the performance of the threshold policies on the three representative phone calls. "MDP 1" and "MDP 2" correspond to the policies from Tables 6, 7, 8, and "85%, $M = i$" corresponds to a heuristic energy threshold policy with p_{th} in terms of percentage and period of joint synchronization $M = i$. We see from the tables that the 85% threshold policy with $M = 1$ gives results that are very close to the "MDP 1" case, with slight degradation in the average sync period and deviation. For all three call cases, the number of joint synchronizations is the same for these two policies, although the "MDP 1" case does experience one solo email synchronization.

Table 11 A day with heavy voice call (121 minutes)

Metric	MDP 1	MDP 2	85%, $M = 1$	85%, $M = 2$
N_b^s	23	14	23	14
N_1^s	1	14	0	14
$Mean_1$	39.88	34.18	41.43	34.11
Dev_1	116.48	111.04	119.5	110.27
N_2^s	0	0	0	0
$Mean_2$	41.61	68.36	41.43	68.21
Dev_2	118.82	163.48	119.5	156.56
M_{call}	121	121	121	121
T_{out}	N.A.	N.A.	N.A.	N.A.
$E_r(T)$	0.43	0.14	1.52	0.14

The 85% case with prioritization performs quite well in terms of protection of call minutes. Since the joint synchronization period $M = 2$ is imposed, it is expected that the number of times service 1 is synchronized ($N_1^s + N_b^s$) is approximately twice that of the service 2 syncs ($N_2^s + N_b^s$). In all three call cases we see that there is no loss of call minutes, and the energy lasts until the recharge time. For all 13 cases there were negligible differences between the performance of the MDP policies and the threshold policies in terms of call minutes served or energy outage times.

4.2.3 Synthetic call traces

To extensively evaluate the performance of the MDP and energy threshold policy, we generate synthetic traces based on different arrival rates. We generate MDP policies and energy threshold functions based on these synthetic traces, and also observe the robustness of the real call profile based policies in these synthetic call environments. These simulations are primarily meant to test the robustness of these methods to various call loads.

In the simulations, synthetic user call profiles are generated corresponding to a constant probability of call arrival p in each time slot, i.e. for $t \leq T$,

$$p^c(t) = p.$$

We assume that all call lengths are equal to 1.

When generating the MDP policies the simulation parameters are the same as in the previous section. The reward parameters used correspond to the "MDP 1" entry of Tables 6–8. The call reward is $R^c = 20$ and the synchronization reward functions $R_1^s(\tau_1)$, $R_2^s(\tau_2)$ are as stated before with $C_1/C_2 = 2$, $C_1 + C_2 = 1$. For each p, a distinct MDP policy is generated using the same call reward R^c and synchronization reward functions $R_i^s(\tau_i)$ as described above. Distinct energy threshold functions are also generated from the same synthetic profiles, all with a joint synchronization period $M = 1$, so that both services refresh simultaneously.

Each policy's average performance is observed over 1000 instantiations of call days generated from its corresponding synthetic call profile. Figures 7–9 show the performance of these simulations for various metrics. In each figure, the MDP policy is generated for values ranging from .02 to .13. This value is chosen as the maximum because $p = .14$ corresponds to an average greater than 133 minutes in a day, which is the maximum number of call

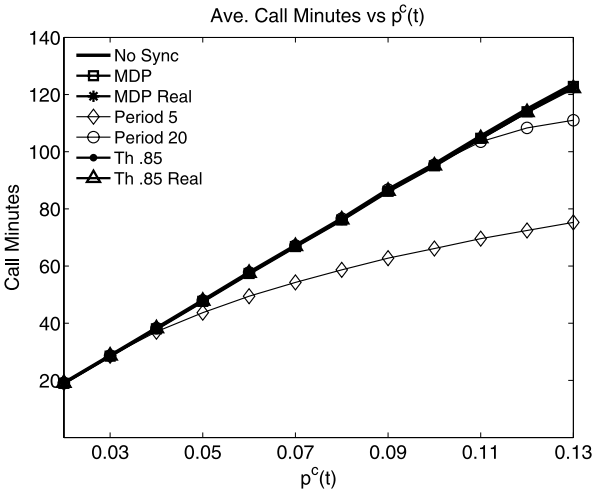

Fig. 7 Average call time vs. probability of call arrival

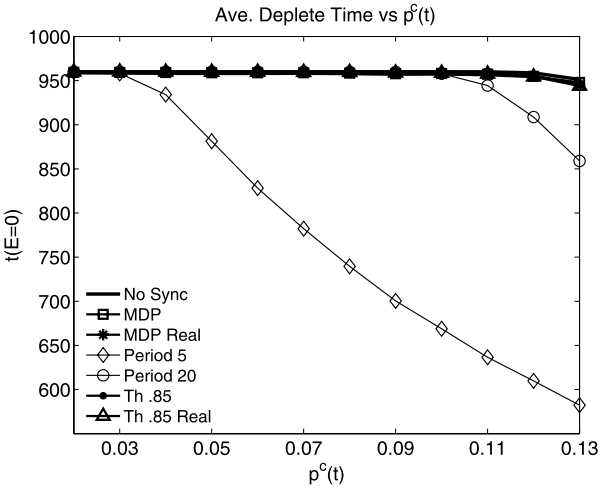

Fig. 8 Average battery depletion time vs. probability of call arrival

minutes achievable with 400 energy units. The performance of the MDP policies generated from the p values is labeled "MDP", and the threshold policies "Th p_{th}", corresponding to different probability threshold values.

Along with the synthetic MDP policies, we show the performance of two fixed-frequency policies. We also take the "MDP 1" policy and the $p_{th} = .85$, $M = 1$ energy threshold policy obtained from the real call traces in the previous sections, and show their performance over the synthetic call days (labeled "MDP real" and "Th .85 Real" respectively). In the "Real" curves, a *single* policy that is generated based on the real trace is run over the entire range of $p^c(t)$. The purpose of this comparison is to evaluate the robustness of the proposed policies against inaccurate information on call arrival distributions. Finally, each figure also shows these metrics when no synchronization occurs (labeled "No Sync"), which serves as the baseline for the performance metrics.

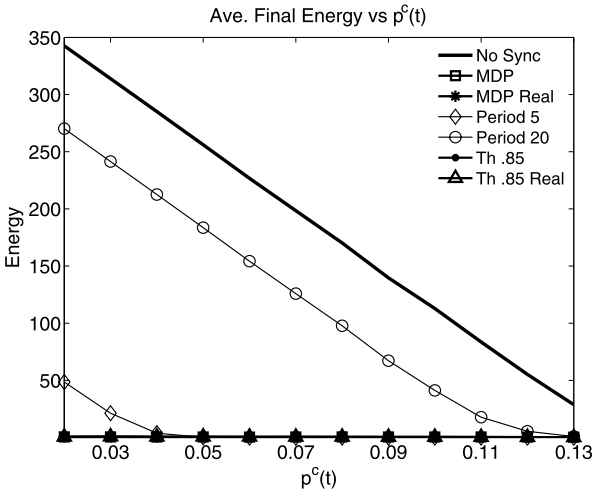

Fig. 9 Average remaining energy at T vs. probability of call arrival

Figure 7 shows the average number of call minutes achieved by the policies listed for various call arrival rates. For this metric, a strong performing MDP policy is one where the average call minutes achieved under the MDP policy approaches the No Sync case. The MDP policies generated from the synthetic call profiles, represented by the "MDP" curve, track the No Sync call load closely for the entire range of $p^c(t)$. In fact, these MDP policies on average achieve within 1 minute of the No Sync case, except for $p^c(t) = .13$, where the gap is roughly 1.5 minutes. This gap is due to the fact that in this probability range, the number of call minutes in a day is more likely to approach or surpass the maximum call time that can be served by the initial energy (133 minutes). The threshold policy (labeled "Th .85") likewise performs well in terms of the number of call minutes achieved. This result closely mirrors our real case result, where we saw that the energy threshold performed very similarly to the MDP results, with small deviations in the refresh period and standard deviation.

We also see that the MDP and energy threshold policies from real call data (labeled "real") also adapt quite well to these different call profiles. In these curves, the "MDP 1" and "85%, $M = 1$" policies from the previous section are run over the entire range of $p^c(t)$. This shows that the MDP and threshold policies are robust to different call loads, refreshing more frequently when call load is lower. The fixed-frequency policies all diverge from the No Sync case earlier than the MDP, with the shorter synchronization periods diverging in lower p values. From this figure we conclude that the MDP and energy threshold policies generated sufficiently protect call time over the entire range of p.

Figure 8 shows the time of energy depletion as a function of the arrival probability. In this case, again, the MDP policy performance tracks the No Sync time horizon line closely. Given that the call time is well protected by both the MDP and energy threshold cases, it follows that the average battery lifetime should be close to the time horizon. There is a slight drop for $p > .11$, which occurs because the number of call minutes is more likely to go over the maximum call time achievable at higher p values, resulting in battery depletion before the recharge time for even the No Sync case. The real trace MDP and energy threshold policies also follow the same trend, achieving slightly lower energy depletion times on average. The fixed-frequency policies follow the same trends as before, with higher frequency policies depleting the battery earlier.

Finally, Fig. 9 shows the energy remaining for each policy. The dynamic policies, on average, achieve near-zero remaining energy, and from Fig. 8 it is clear that these policies are not prematurely draining the battery. The fixed-frequency policies do not deplete their energy for lower call arrival values, but deplete the battery prematurely in heavy load cases.

These results show that both the MDP and energy threshold policies result in strong performance over a wide range of call loads. This means that on days with wide-ranging call loads, the same policies can adapt synchronizations in a way that will still protect voice activity. However, this study does not necessarily indicate strong performance in the face of more bursty call profiles. For example, if a user has extremely sparse call arrival but tends to talk for long periods when calls do arrive, MDP may not be sensitive enough to protect these unlikely events. Further study should be done to see how MDP policies can be modified to perform well in these situations.

4.3 Discussion

Next we discuss the computational complexity and scalability of the proposed approaches. First, we have the option of implementing the computationally expensive algorithm on the phone or on a more powerful server through an Internet connection. For instance, to generate the decision table used in email synchronization, it takes less than a second on a desktop computer and one hour on the G1 phone we used. The delay is mainly due to the memory constraint on the cellular phone. The decision table does not have to be computed frequently (once every few weeks) since call profiles do not change significantly over small time scales. As was shown in Sect. 4.2.3, a single MDP-based policy can be robust for various call arrival rates.

Theorem 1 allows us to reduce the decision table size by utilizing the structure of the solution. For the case presented in the paper, the table size is reduced from 480 (# of time units) $*$ 100 (power level) $*$ 480 (max sync delay) $*$ 1 bit to 480 (# of time units) $*$ 100 (power level) $*$ 1 bit. In other words, the size is reduced from 2.88 MB to 48 KB.

Another method to improve scalability is to reduce granularity. In earlier discussions, we consider the case where each time slot is 2 minutes. We also evaluate the cases where the time slots are 4, 6, 8 and 16 minutes, where the table sizes are $1/2$, $1/3$, $1/4$ and $1/8$ of the original. Energy granularity reductions are also possible. Although the MDP formulation assumes integral values for all state variables, non-integral values can be used through simple modification of the MDP state transition probabilities. The performance degradation is minor and graceful.

For several data synchronization services, scalability becomes difficult in the MDP formulation, even with special structure and granularity reductions. The energy threshold heuristic has also been shown to perform well in our results, and is easily scalable to multiple applications.

Non-zero reward for remaining energy could be explored also. We considered a non-zero reward function, $f(E_r) = c \log(1 + E_r)$, where c is a constant. The main benefit is to reduce the variance of data synchronization, especially when the time is close to the charging time. We also considered different reward functions, including various logarithm and square-root functions. The impact is somewhat minor.

5 Related work

The work described in this paper is broadly related to the general problem of *adapting* and *managing* resources at the system level. As a result, there is work related to this in many

disciplines such as operating systems, real-time systems, computer architecture, networking, and more recently in sensor networks and mobile computing.

Stanford researchers [6] were among the first to use Markov decision processes to address power optimization policies for notebook or other battery-operated systems. This work is primarily concerned with maximizing battery life. Our work differs because it is more dynamic in nature, since our goal is to maximize user experience until the expected charging time. For example, in [6], the optimal action to turn on/off a disk does not change over time, but in this work optimal decisions depend explicitly on the current time and remaining energy. Quality versus resource utilization trade-offs have also been studied extensively in the area of video streaming [3, 9, 14, 15, 18–20]. This is somewhat orthogonal to our work, since most streaming applications would be user-initiated and therefore not delay tolerant.

In [17] researchers from Intel and Microsoft propose the idea of context-aware battery management and the notion of treating the next recharge time explicitly. However, the paper does not address the issue of controlling applications to preserve battery life until an expected recharge time, which is the focus of this paper.

CMU researchers studied OS support for resource scalable computation and energy aware adaptive computation [9, 16, 18]. In particular, in [9] the authors demonstrate a 30% extension in battery life through collaborative optimization of the operating systems and the application. Duke researchers [21] extend this approach to the system level by formulating a general framework to manage energy as a first class operating system resource. They propose a currency model to account for energy consumed by different components and develop techniques for fair allocation of available energy to *all* active applications. In [8], the authors propose a dynamic software management framework to improve battery life that is based on quality-of-service (QoS) adaptation and user-defined priority. The goal of their works is to extend the battery lifetime by limiting the average discharge rate. All of their works focus on a general purpose notebook computer with no particular priority for different functions, whereas our work is specifically concerned with mobile phones with different priority services.

In the area of sensor networks, UCLA researchers [13] discuss scheduling tasks to accommodate the constraints of energy harvested from the environment such as solar panels. In [4], an alternative approach to reactive optimization is discussed. The main difference between these works and our own is in the MDP formulation of the talk-time optimization problem in the context of mobile phones and its implementation on the Android powered mobile phone.

6 Conclusions

In this paper we proposed a general mathematical framework to optimize energy consumption on mobile phones using the Markov decision process. We argued that on a mobile phone, certain functions have priority to the user, and the optimization's primary goal should be to protect these functions. Therefore, the usage profile of these functions should be an input to the optimization framework, since the optimization would depend on the usage patterns of these high priority functions. This made the optimization user profile driven and adaptive, as the user profile is essentially a user-defined parameter that varies from one individual to another.

In our formulations, we used talk-time as the primary function to be preserved, and used call histories as an input to the optimization problem, with the caveat that other functions could be similarly modeled and input as primary phone functions. We first demonstrated

the MDP framework on a single delay tolerant function, email, and developed techniques to reduce the table size of the MDP decision table for this application. To address the problem of multiple data synchronization functions, we designed an energy threshold heuristic that performs well and is highly scalable as the number of applications climbs. We tested the MDP framework on both a single data synchronization function and multiple data synchronization functions, and showed through real trace simulations that the framework was robust enough to handle both situations while sufficiently protecting call-time and battery life. We also tested the heuristic for a multiple data synchronization function, and showed that the results were similar to the MDP framework approach. Finally, we simulated over several synthetic call profiles to show the robustness of our methodology for various types of user profiles.

Acknowledgement This work supported in part by NSF CNS-0435531, CNS-0448613 and CNS-0520126 and by Intel through a gift grant.

Appendix A: Proof of Theorem 1

Property 1 *Given t and E_r, the following property is true*:

$$V(t, E_r, \tau + \epsilon) - V(t, E_r, \tau) \leq R^s(\tau + \epsilon) - R^s(\epsilon), \tag{10}$$

for $\epsilon \geq 0$.

Proof The property can proved using backward induction. Because of the boundary condition in (2), we have

$$\begin{aligned} & V(T, E_r, \tau + \epsilon) - V(T, E_r, \tau) \\ & \quad = f_r(E_r) - f_r(E_r) \\ & \quad = 0 \\ & \quad \leq R^s(\tau + \epsilon) - R^s(\epsilon). \end{aligned}$$

Therefore, (10) holds for $t = T$. We next use backward induction. Assume (10) holds for $t + 1, t + 2, \ldots, T$. We now prove it holds for t.

Consider $V(t, E_r, \tau + \epsilon) - V(t, E_r, \tau)$. We have

$$\begin{aligned} & V(t, E_r, \tau + \epsilon) - V(t, E_r, \tau) \\ & \quad = p^c(t)\left(v^c(t, E_r, \tau + \epsilon) - v^c(t, E_r, \tau)\right) \\ & \qquad + (1 - p^c(t)) \max\left(v^s(t, E_r, \tau + \epsilon), v^i(t, E_r, \tau + \epsilon)\right) \\ & \qquad - (1 - p^c(t)) \max\left(v^s(t, E_r, \tau), v^i(t, E_r, \tau)\right) \\ & \quad \overset{(1)}{\leq} p^c(t)\left(v^c(t, E_r, \tau + \epsilon) - v^c(t, E_r, \tau)\right) \\ & \qquad + (1 - p^c(t)) \max\{v^s(t, E_r, \tau + \epsilon) - v^s(t, E_r, \tau), v^i(t, E_r, \tau + \epsilon) - v^i(t, E_r, \tau)\}, \end{aligned}$$

where (1) holds because

$$\begin{aligned}&\max(a,b)-\max(c,d)\\&\quad=\max\left(a-\max(c,d),b-\max(c,d)\right)\\&\quad\le\max(a-c,b-d).\end{aligned}$$

We consider the first term next.

$$\begin{aligned}&v^c(t,E_r,\tau+\epsilon)-v^c(t,E_r,\tau)\\&\quad=E_{L^c}\left[V(t+L^c,(E_r-L^c*e^c)^+,\tau+\epsilon+L^c)\right.\\&\qquad\left.-V(t+L^c,(E_r-L^c*e^c)^+,\tau+L^c)\right]\\&\quad\overset{(2)}{\le}E_{L^c}\left[R^s(\tau+\epsilon+L^c)-R^s(\tau+L^c)\right]\\&\quad\overset{(3)}{\le}E_{L^c}\left[R^s(\tau+\epsilon)-R^s(\tau)\right]\\&\quad=R^s(\tau+\epsilon)-R^s(\tau)\end{aligned}$$

where (2) holds by the hypothesis and (3) holds because $R^s(\cdot)$ is a subadditive increasing function. Consider the second term.

$$\begin{aligned}&\max\left\{v^c(t,E_r,\tau+\epsilon)-v^c(t,E_r,\tau),v^i(t,E_r,\tau+\epsilon)-v^i(t,E_r,\tau)\right\}\\&\le\max\left\{V(t+1,(E_r-e^s)^+,1)+R^s(\tau+\epsilon)\mathbf{1}\{E_r\ge e^s\}\right.\\&\quad-V(t+1,(E_r-e^s)^+,1)+R^s(\tau)\mathbf{1}\{E_r\ge e^s\},\\&\quad\left.V(t+1,E_r,\tau+\epsilon+1)-V(t+1,E_r,\tau+1)\right\}\\&\le\max\left\{R^s(\tau+\epsilon)-R^s(\tau),R^s(\tau+\epsilon+1)-R^s(\tau+1)\right\}\\&\le R^s(\tau+\epsilon)-R^s(\tau).\end{aligned}$$

Combining the above two results, we have

$$V(t,E_r,\tau+\epsilon)-V(t,E_r,\tau)\le R^s(\tau+\epsilon)-R^s(\tau).\qquad\square$$

Based on the above property, we prove Theorem 1 next.

Proof of Theorem 1 Given (t,E_r), we need to prove that $\forall\ \tau\ge\tau^*(t,E_r)$, we have

$$v^s(t,E_r,\tau)\ge v^i(t,E_r,\tau).\tag{11}$$

If $E_r<e^s$, then

$$v^s(\tau)=V(t+1,(E_r-e^s)^+,1)+R^s(\tau)\mathbf{1}\{E_r\ge e^s\}=0.$$

Therefore, the optimal action is to stay idle. In this case, set $\tau^*=\infty$. The result is trivial. So we only consider the case $E_r\ge e^s$ in the following

$$\begin{aligned} v^s(t, E_r, \tau) &= V(t+1, E_r - e^s, 1) + R^s(\tau) \\ &= V(t+1, E_r - e^s, 1) + R^s(\tau^*) + R^s(\tau) - R^s(\tau^*) \\ &= v^s(\tau^*) + R^s(\tau) - R^s(\tau^*) \\ &\overset{(4)}{\geq} v^i(\tau^*) + R^s(\tau) - R^s(\tau^*). \end{aligned}$$

In the above, (4) holds by the definition of τ^*

$$\begin{aligned} v^s(t, E_r, \tau) &= V(t+1, E_r, \tau+1) \\ &\overset{(6)}{\leq} V(t+1, E_r, \tau^*+1) + R^s(\tau+1) - R^s(\tau^*+1) \\ &= v^i(\tau) + R^s(\tau+1) - R^s(\tau^*+1) \\ &\overset{(7)}{\leq} v^i(\tau^*) + R^s(\tau) - R^s(\tau^*). \end{aligned}$$

In the above, (6) holds by Property 1 and (7) holds by concavity of $R^s(\cdot)$. Therefore, we have

$$v^s(t, E_r, \tau) \geq v^i(t, E_r, \tau)$$

for $\tau \geq \tau^*$. □

Appendix B: Derivation of email/twitter application MDP

We now derive the value function in the Email/Twitter Application. Recall that the system state in this application is represented by the tuple (t, E_r, τ_1, τ_2). We begin by defining $v^c(\cdot)$ and $v^i(\cdot)$,

$$\begin{aligned} v^c(t, E_r, \tau_1, \tau_2) = E_{L^c}\Bigg[& V(t + L^c, (E_r - L^c * e^c)^+, \tau_1 + L^c, \tau_2 + L^c) \\ & + \min\left(\left\lfloor \frac{E_r}{e^c} \right\rfloor, L^c\right) R^c \Bigg], \end{aligned} \tag{12}$$

$$v^i(t, E_r, \tau) = V(t+1, E_r, \tau_1 + 1, \tau_2 + 1), \tag{13}$$

where $x^+ = \max(0, x)$. These equations are similar to the single-application case. The same goes for the equations for individual synchronization, $v_1^s(\cdot)$ and $v_2^s(\cdot)$,

$$v_1^s(t, E_r, \tau_1, \tau_2) = V\left(t+1, (E_r - e_1^s)^+, 1, \tau_2 + 1\right) + R_1^s(\tau_1)\mathbf{1}\{E_r \geq e_1^s\}, \tag{14}$$

$$v_2^s(t, E_r, \tau_1, \tau_2) = V\left(t+1, (E_r - e_2^s)^+, \tau_1 + 1, 1\right) + R_2^s(\tau_2)\mathbf{1}\{E_r \geq e_2^s\}, \tag{15}$$

where in each case, we see that τ_i is reset to 1 when that service is synched. Finally, for synching both services together, we have $v_b^s(\cdot)$ as

$$v_b^s(t, E_r, \tau_1, \tau_2) = V\left(t+1, (E_r - e_b^s)^+, 1, 1\right) + \left(R_1^s(\tau_1) + R_2^s(\tau_2)\right)\mathbf{1}\{E_r \geq e_b^s\}. \tag{16}$$

For synchronizing both services together, we receive the reward for both services. We also note that the energy cost here is e_b^s, which is the energy cost for synchronizing both services simultaneously. Using these value functions, we can now obtain (4) and (5), the optimality equation and initial conditions for the MDP.

Appendix C: Sample email envelope

```
Delivered-To: XXXXXX@gmail.com
Received: by 10.229.83.135 with SMTP id f7cs43124qcl;
        Tue, 5 Jan 2010 14:07:02 -0800 (PST)
Received: by 10.224.98.34 with SMTP id o34mr87019qan.327.1262729222170;
        Tue, 05 Jan 2010 14:07:02 -0800 (PST)
Return-Path: <sender@email.com>
Received: from mail-qy0-f184.google.com (mail-qy0-f184.google.com [209.85.221.184])
        by mx.google.com with ESMTP id 16si26880281qyk.15.2010.01.05.14.07.01;
        Tue, 05 Jan 2010 14:07:01 -0800 (PST)
Received-SPF: pass (google.com: domain of sender@email.com designates
209.85.221.184 as permitted sender) client-ip=209.85.221.184;
Authentication-Results: mx.google.com; spf=pass (google.com: domain of
sender@email.com designates
209.85.221.184 as permitted sender) smtp.mail=sender@email.com; dkim=pass
(test mode) header.i=@gmail.com
Received: by mail-qy0-f184.google.com with SMTP id 14so6988382qyk.11
        for <ucdmcsg@gmail.com>; Tue, 05 Jan 2010 14:07:01 -0800 (PST)
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
        d=gmail.com; s=gamma;
        h=domainkey-signature:received:received:message-id:date:from
         :user-agent:mime-version:to:subject:content-type
         :content-transfer-encoding;
        bh=Pg7uu0WObNWgjgx2hQHxGxsY0sy3f8gP/kBF+yYlr54=;
        b=QHJf3qKn+Yq9s176eDML2UvaQkAsFC5YHMgRWWE5q+vkX8iMu4Y+reqSSOPNyxmeMV
         SxVkG8xVLWInqEa2dsSNpXUgqzxp1AEctMYrY4Gh9pk3rjHGiumWxEwAhvN5SZLi5Uyb
         J3OnRes6F2afae37z9FDE1PhDGpK1+x75GVd0=
DomainKey-Signature: a=rsa-sha1; c=nofws;
        d=gmail.com; s=gamma;
        h=message-id:date:from:user-agent:mime-version:to:subject
         :content-type:content-transfer-encoding;
        b=PF/T/SanmAgXWh3KlXhoqjJxxuusD6Zk31u4MNcZTUb7bkiB/Ac0zUeUC12CXIfK22
         QSiQPfcB+5X0jh4xgdOmyLIkK+fgZ8TLX+BH5hbFeF48yGiZv1AIKagT5Bnb2qBqz3A9
         /+qAyShpKs9UrkXBgjT2nHbseS6tvDoJIn/z0=
Received: by 10.224.88.75 with SMTP id z11mr4826971qal.70.1262729221044;
        Tue, 05 Jan 2010 14:07:01 -0800 (PST)
Return-Path: <sender@email.com>
Received: from dhcp-169-237-152-78.ece.ucdavis.edu ([169.237.152.78])
        by mx.google.com with ESMTPS id 22sm17646587qyk.6.2010.01.05.14.06.59
        (version=SSLv3 cipher=RC4-MD5);
        Tue, 05 Jan 2010 14:07:00 -0800 (PST)
Message-ID: <4B43B802.7090206@gmail.com>
Date: Tue, 05 Jan 2010 14:06:58 -0800
From: Sender <sender@email.com>
User-Agent: Postbox 1.1.0 (Macintosh/20091201)
MIME-Version: 1.0
To: XXXXXX@gmail.com
Subject: Test #10
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 7bit
```

References

1. Javamail port for the android plateform. http://code.google.com/p/javamail-android/
2. Twitter4j—a java library for the twitter api. http://yusuke.homeip.net/twitter4j/en/index.html
3. Akella V, van der Schaar M, Kao W-F (2005) Proactive energy optimization algorithms for wavelet-based video codecs on power-aware processors. In: IEEE international conference on multimedia and Expo, pp 566–569
4. Alur R, Kanade A, Weiss G (2008) Ranking automata and games for prioritized requirements. In: 20th international conference on computer-aided verification
5. Apple. Apple's app store downloads top three billion. Press Release, January 2010. http://www.apple.com/pr/library/2010/01/05appstore.html
6. Benini L, Bogliolo A, Paleologo GA, Micheli GD (1998) Policy optimization for dynamic power management. IEEE Trans Comput Aided Des Integr Circuits Syst 18:813–833

7. Bronger T Python gpib etc. support with pyvisa, controlling gpib, rs232, and usb instruments. http://pyvisa.sourceforge.net/
8. Fei Y, Zhong L, Jha NK (2008) An energy-aware framework for dynamic software management in mobile computing systems. ACM Trans Embed Comput Syst 7(3):1–31
9. Flinn J, Satyanarayanan M (2004) Managing battery lifetime with energy-aware adaptation. ACM Trans Comput Syst 22(2):137–179
10. Google. Android, official website, April 2009. http://www.android.com/
11. Google. Google offers new model for consumers to buy a mobile phone. Press Release, January 2010. http://www.google.com/intl/en/press/pressrel/20100105_phone.html
12. HTC. Htc g1 overview. http://www.htc.com/www/product/g1/overview.html
13. Kansal A, Potter D, Srivastava M (2004) Performance aware tasking for environmentally powered sensor networks. SIGMETRICS Perform Eval Rev 32(1):223–234
14. Mohapatra S, Cornea R, Dutt N, Nicolau A, Venkatasubramanian N (2003) Integrated power management for video streaming to mobile handheld devices. In: MULTIMEDIA'03: proceedings of the eleventh ACM international conference on multimedia. ACM, New York, pp 582–591
15. Mohapatra S, Cornea R, Oh H, Lee K, Kim M, Dutt N, Gupta R, Nicolau A, Shukla S, Venkatasubramanian N (2005) A cross-layer approach for power-performance optimization in distributed mobile systems. In: IPDPS'05: proceedings of the 19th IEEE international parallel and distributed processing symposium (IPDPS'05)—workshop 10. IEEE Comput Soc, Los Alamitos, p 218.1
16. Narayanan D, Satyanarayanan M (2003) Predictive resource management for wearable computing. In: MobiSys'03: Proceedings of the 1st international conference on mobile systems, applications and services. ACM, New York, pp 113–128
17. Ravi N, Scott J, Han L, Iftode L (2008) Context-aware battery management for mobile phones. In: Sixth annual IEEE international conference on pervasive computing and communications, pp 224–233
18. Satyanarayanan M, Narayanan D (2001) Multi-fidelity algorithms for interactive mobile applications. Wirel Netw 7(6):601–607
19. van der Schaar M, Turaga D, Akella V (2004) Rate-distortion-complexity adaptive video compression and streaming. In: International conference on image processing, ICIP'04, vol 3, pp 2051–2054
20. Wanghong Y, Nahrstedt K, Sarita Adve DJ, Kravets RK (2006) Grace-1: Cross-layer adaptation for multimedia quality and battery energy. IEEE Trans Mob Comput 5(7):799–815
21. Zeng H, Ellis CS, Lebeck AR, Vahdat A (2002) Ecosystem: managing energy as a first class operating system resource. ASPLOS 37(10):123–132