

Markov Decision Process (MDP) Framework for Optimizing Software on Mobile Phones

Tang Lung Cheung
Department of Computer
Science
University of California, Davis
Davis, California
ctlcheung@ucdavis.edu

Kari Okamoto
Department of Computer
Science
University of California, Davis
Davis, California
kdokamoto@cs.ucdavis.edu

Frank Maker III
Department of Electrical and
Computer Engineering
University of California, Davis
Davis, California
flmaker@ucdavis.edu

Xin Liu
Department of Computer
Science
University of California, Davis
Davis, California
liu@cs.ucdavis.edu

Venkatesh Akella
Department of Electrical and
Computer Engineering
University of California, Davis
Davis, California
akella@ucdavis.edu

ABSTRACT

We present a framework based on Markov decision process to optimize software on mobile phones. Unlike previous approaches in literature that focus on energy optimization while meeting a specific task-related time constraint, we model the desired talk-time as an explicit user given parameter and formulate the optimization of resources such as battery-life on a mobile phone as a decision processes that maximizes a user specified application specific reward or utility metric while meeting the talk-time constraint. We propose efficient techniques to solve the optimization problem based on dynamic programming and illustrate how it can be used in the context of realistic applications such as WiFi radio power optimization and email synchronization. We present a design methodology to use the proposed technique and experimental results using the Android platform from Google running on the HTC mobile phone.

Categories and Subject Descriptors

D.4.7 [Software]: Operating Systems[Organization and Design[Real-timed and embedded systems]]; D.m [Software]: Miscellaneous

General Terms

Algorithms, Design

Keywords

User-profile driven, Power optimization, Talk time extension, Markov-Decision Process, Android, Mobile Phones

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EMSOFT'09, October 12–16, 2009, Grenoble, France.

Copyright 2009 ACM 978-1-60558-627-4/09/10 ...\$10.00.

1. INTRODUCTION

Mobile phones are ubiquitous embedded systems. With every generation their computing capability is growing. On April 21, 2009, Apple reported that over a billion iPhone applications were downloaded in just over a year. In the past year Google introduced a new operating system and application development kit called Android for mobile handsets; Microsoft and Nokia are responding with similar initiatives this year. What this means is that, just like the PC was the target platform for software developers in the 80's and 90's, the mobile phone is the platform for the next decade and beyond. So, we need software design and optimization techniques to support this new platform.

A mobile phone has key differences when compared to a desktop or a notebook PC. First, a mobile phone is an *embedded system* as opposed to a general-purpose computer. Its primary function (to most users) is voice communication. Second, mobile phones are more constrained than PCs (including notebooks, netbooks etc) in terms of their form factor and weight. Therefore, applications have to be more conscious about their resource usage, especially memory and battery. Third, the nature of mobile phones is more event-driven or *reactive*, which means the same application is launched many times and the applications are active for a short duration (e.g., minutes instead of hours or days). Last, a mobile phone typically has exactly one user, unlike a PC which might be shared. So, there is a great opportunity to *customize* the application to the usage pattern, which could be very diverse from one user to another. For example, different people might have different preferences of when they charge their cellphones, and different quality of access to WiFi and cellular radio coverage, etc. So, a *one-size-fits-all* approach to application development, which is prevalent in the PC environment, is not appropriate for developing embedded software for mobile phones.

We present a technique for developing and *optimizing* applications to address these concerns. More specifically, we propose a Markov Decision Process (MDP) based framework for dynamic optimization of applications running on a mobile phone. The MDP framework allows the applications to incorporate user preference and user profiles into *decisions*

making at run time about what resources to use, at any given time such that some utility or reward function is optimized.

In this paper we focus on power consumption of an application, which in turn affects the battery-life and hence the talk-time of the mobile phone. We believe power consumption is important because (as noted above) the primary purpose of a mobile phone is to provide voice communication. So, it is important that *other* applications such as email, browser, games, and utilities/tools that users run on their mobile phones do not consume too much power to disrupt the primary function of the phone. That we believe is a key difference of optimizing power for applications running on a mobile phone versus the general software power optimization. Our problem formulation will explicitly take into account the desired talk time as an input parameter. This will be modeled as the time T till which we expect the battery to last, with the underlying assumption that the phone will be recharged at that time. The optimizer will try to maximize the utility (reward) of using different applications while making the battery last till time T . This is a subtle but important difference between the proposed work and the traditional problem of optimizing energy on embedded systems that primary focus on maximizing the battery-life by dialing down performance via dynamic voltage and frequency scaling to meet a real-time constraint. For example in [?, ?, ?, ?] researchers present a cross-layer optimization methodology for video decoding, by dynamically scaling the voltage and frequency of the underlying processor such that a task of decoding a given video frame finishes *just in time*. The time constraint could be predicted based on the time required to process a frame derived on the previous history. Though there is a time constraint in both cases, in our cases it is a macro-level (global) constraint that captures the need to preserve the capability to talk till the battery gets recharged, instead of a *local* time constraint to process a frame of video sequence or complete a given task. The time constraint in our case is global in the sense that it applies to all tasks and applications that run between now and the user specified time T . In addition, the time constraint in our case is specified by the user, i.e. it is an external (input) parameter, not based on the workload estimation, as in almost all papers related to this work in the literature. In addition, the parameter T , assumed given in the current paper, is user-profile driven and can be estimated with good accuracy as shown in [?].

In summary, the key contributions of this paper are -

1. A methodology for *dynamic* power optimization of applications to prolong the life time of a mobile phone till a user specified time while maximizing a user defined reward function.
2. A mathematical formulation of the problem via Markov decision processes and techniques to reduce the size of the decision tables.
3. An illustration of using the technique on two applications based on the Android software development platform.

The rest of the paper is organized as follows. In Section 2 we will present the problem formulation. We will first introduce the general concept of MDP, and then develop mathematical formulations using two case studies. We will also

present techniques to solve the mathematical formulation efficiently. In Section 3 we present the experimental setup based on the Android platform and show how the key parameters are estimated from the mobile phone hardware. In Section 4 we will present results from two case studies that use the proposed optimization framework. These sections will also describe how the user profile data is generated and used within the framework and how the optimization procedure is used in conjunction with other applications. In Section 5 we review related work from literature and in Section 6 we will summarize the key ideas in the paper and present directions for future work.

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 high priority service and other delay-tolerant data applications as low priority service. Our objective is to minimize disruption to voice communication due to power depletion caused by other lower priority applications. Intuitively, when the remaining battery is sufficient with respect to expected charging time, one can run other applications 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, one should conserve energy for other lower priority application to maintain availability for voice communications. Information related to charging time, voice communication pattern, etc. are user profile driven.

We study two generic tasks that are useful to a wide variety of applications on mobile phones - Data synchronization and WiFi radio control. Data synchronization represents refreshing content (data) from a remote server. An example of it is email synchronization. WiFi radios consume a significant amount of power on mobile phone, so deciding when to wake up the WiFi radio and when to turn it off can have significant implications on the battery-life and hence on the talk-time of the phone.

2.1 Markov Decision Process

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

S is the state space, $s \in S$ is the current state, known to users. In this paper, the state includes the current time and remaining energy (battery-life) (and some others).

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 to synchronize email or not, or to turn the WiFi radio on or off.

$P_a(s)$ 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 (e.g., 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, the immediate reward is zero.

The objective in this paper is to maximize the cumulative reward until battery charging time. We propose to use the

MDP approach to better handle the dynamics of the system because phonecalls are stochastic. The key of the proposed approach is to utilize energy resource dynamically.

In this case, an optimal action apparently 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 now yields an immediate reward, at the cost of energy consumption, which may reduce its future reward. All these factors need to be considered in the decision.

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 (namely, the mobile phone), so it will not be useful if it takes up 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 will show that in some cases, structures exist so that the optimal policy has a simpler format (e.g., a threshold-based format), which can be exploited to reduce the computational complexity as well as the space required to store the optimal solution.

In our case, the number of actions is limited, and the time horizon is finite. Time is slotted. Each time slot is a time unit. The decision is made at the beginning of the time slot given the information on the current state. In the data synchronization example, a time slot is two minutes. We can also consider a larger time unit to reduce complexity at the tradeoff of a coarse granularity.

2.2 MDP Model for Data Synchronization

We will consider data synchronization applications that are delay tolerant - examples of this include email, calendar, contacts, refreshing facebook pages etc. Intuitively, if the phone is close to its expected charging time and has abundant energy left, we could run the data synchronization applications more often. On the other hand, if the charging time is far way in the future, one 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 use 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 : unit time power consumption of voice call
- L_c : length of a voice call, a random variable
- R_c : reward of one unit of voice call
- $p_c(t)$: Prob(voice call arrives in a time unit)
- $R_s(\tau)$: reward of mail synchronization, subadditive
- e_s : Energy consumption 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. In other words,

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

Last, $x^+ = \max(0, x)$ (indicating that remaining energy cannot be negative in this paper).

We assume that $R_s(\cdot)$ is an increasing subadditive function. It is increasing because the longer the delay, the larger the need and thus the higher the reward of synchronization. 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., 10pm). It could also be dynamic, obtained based on user profiling.

We use the MDP framework. In this framework, 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 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.

$$\begin{aligned} v_c(t, E_r, \tau) &= E_{L_c} \left[V(t + L_c, (E_r - L_c * e_c)^+, \tau + L_c) \right. \\ &\quad \left. + \min \left(\frac{E_r}{e_c}, L_c \right) R_c \right] \\ v_s(t, E_r, \tau) &= V(t + 1, (E_r - e_s)^+, 1) + R_s(\tau) \mathbf{1}\{E_r \geq e_s\}, \\ v_i(t, E_r, \tau) &= V(t + 1, E_r, \tau + 1) \end{aligned}$$

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 $\min(E_r/e_c, L_c) R_c$, which is proportional to the length of the phone call, supported by the battery. We assume that when a phone call occurs, no synchronization activity is allowed. The terms v_s and v_i correspond to the case when no phone call happens in this time slot. In this case, the value is v_s if the action is to synchronize mail and the value is v_i 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:

$$\begin{aligned} V(t, E_r, \tau) &= p_c(t) v_c(t, E_r, \tau) + \\ &\quad (1 - p_c(t)) \max \{v_s(t, E_r, \tau), v_i(t, E_r, \tau)\} \mathbf{1} \end{aligned}$$

In other words, when no phone call occurs, the battery manager 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 optimal decision for each (t, E_r, τ)) using dynamic

programming through backward reduction¹. We have the following boundary conditions.

$$V(T, E_r, \tau) = f_r(E_r) \quad (2)$$

$$V(t, 0, \tau) = 0 \quad (3)$$

To be more specific, the boundary condition defines the optimal decision at time T , and when the phone is out of battery. The first equation evaluates 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 remained at the charging time has no value. Note that remaining energy may have values, especially if the charging time is not a constant. Given the boundary condition at time T , one can then find the optimal action at time $T - 1$. Using backward induction, if we have the optimal decision (and the corresponding value) at time $t + 1, t + 2, \dots, T$, one can use the optimality equation to find the optimal decision at time t . In this approach, the solutions can be represented using a three-dimension table; i.e., one 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. In general this could be very large, especially if the granularity of the time at which the optimization is made 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*

$$\tau^*(t, E_r) = \min(\tau : v_s(\tau) \geq v_i(\tau)),$$

The following policy is optimal:

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

The proof of the theorem is presented in the Appendix. The structure is useful in reducing the complexity/memory space required for the optimal policy. Instead of a three-dimension table, one can represent the optimal policy using a two dimensional one, i.e., $\tau^*(t, E_r)$, as shown in the following.

In Table 1, we show a piece of a decision table derived using the measurement data, discussed in detail later. We see that, at time=77 with remaining energy=395 units, the threshold is 48. In other words, if the email has not been synchronized for 48 unit time or more, it should synchronize in this time unit. Otherwise, it should not synchronize. Using the threshold-based policy, the size of the decision table is reduced to the product of length of the discharging period and the number of different energy levels.

2.3 MDP Model for WiFi Interface Control

Most cellular phones have multiple wireless interfaces, such as cellular, WiFi, and Bluetooth. Bluetooth, due to its limited transmission range, is often exclusively used for wireless headset. For data communication, both cellular interface and WiFi interface are viable options. When available,

¹Dynamic programming through backward reduction is 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 early slot depends on the action of a later slot.

Table 1: Threshold-based decision table

| time | E_r | Threshold |
|------|-------|-----------|
| 77 | 396 | 48 |
| 77 | 397 | 47 |
| 77 | 398 | 46 |
| 77 | 399 | 45 |
| 77 | 400 | 45 |
| 78 | 0 | NEVER |
| 78 | 1 | NEVER |

WiFi interface can transmit data at higher data rate, and thus result in lower energy consumption per unit data, as validated in our measurement study in Section 3. In this section, we consider WiFi interface control with delay-tolerant data applications.

WiFi interface can be controlled using the same MDP framework. WiFi interface has the following power consumption characteristics. It takes seconds or tens of seconds to turn on a WiFi interface (which includes scanning for available AP, Associate with AP, obtain IP addresses, etc). During the process, it consumes a fair amount of energy. When the WiFi radio is on, compared to the cellular interface, energy consumption per unit data is low. When the WiFi radio is on, even if it is not transmitting, it consumes a large amount of energy being idle. So intuitively, one should turn on WiFi radio only when there is a significant amount of data to transmit. When energy is abundant, one can turn on the WiFi interface more frequently to reduce delay. If energy is scarce, one should aggregate more data before turning on the interface. Similar issues exist on when to turn off the WiFi radio.

Because of the long delay involved to turn on/off WiFi radio, we consider a macro-scale WiFi radio interface control that focuses on flows/requests, instead of micro-time scale that focuses on packets. We assume there is a WiFi resource manager that receives requests from different applications that need to transmit data using the WiFi radio. The resource manager will implement the MDP based optimization technique that is the focus of this section. The notion of a transmission request requires some clarification. A transmission request signifies the desire to send a *specific amount* of data continuously. If an application wishes to send more than the specified amount of data, it could make multiple requests to the WiFi resource manager.

We have the following notations in addition to the notations used in the above one.

N : Number of flows queued

$R(N)$: immediate reward of sending N flows

e_w : wakeup energy consumption of WiFi interface

e_m : unit time energy consumption of transmitting one flow

e_I : unit energy consumption of WiFi interface when it is ON

Again, we assume that $R(\cdot)$ is an increasing subadditive function to model the timeliness nature of the reward.

Value function is defined as follows. When the WiFi radio interface is ON, all flows will be transmitted. The decision is whether to turn off WiFi interface after transmission. When the WiFi radio interface is OFF, the decision is whether to turn on the interface. First, consider when the WiFi radio

interface is ON.

$$\begin{aligned}
 V(t, E_r, N, ON) &= R(N) + \\
 \max\{ &E_X [V(t + 1, E_r - Ne_m, X, OFF)], \\
 &E_X [V(t + 1, E_r - Ne_m - e_I, X, ON)]\} \quad (4)
 \end{aligned}$$

where $R(N)$ is the immediate reward of transmitting N flows, the first term inside \max is the total reward to turn off the radio after transmitting N flows, and the second term is that to keep the radio ON. When the radio interface is OFF, we have

$$\begin{aligned}
 V(t, E_r, N, OFF) &= \max\{ \\
 &R(N) + E_X [V(t + 1, E_r - e_w - Ne_m, X, ON)], \\
 &E_X [V(t + 1, E_r, N + X, OFF)]\} \quad (5)
 \end{aligned}$$

where the first term is to turn on the radio, and the second term is to keep the radio off. The optimal policy can be calculated numerically using backward induction.

3. EXPERIMENTAL SETUP

We used the Android Developer Phone 1 (HTC G1 [?]) to obtain power measurement data needed for this paper. We chose the Android platform [?] 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. Also, we were able to remove unnecessary processes during testing to remove their impact on the resulting measurements. This was essential in order to determine the constants necessary to test our proposed framework accurately.

3.1 Device Setup

We measured the power consumption of certain events specific to the Android G1 mobile phone by using a DC power supply (Agilent E3644A [1]) to power the phone instead of the phone’s battery. We connected the power supply to the phone and a computer using the IEEE-488A General Purpose Interface Bus (GPIB). The power measurements were then sampled using a Python script using the PyVISA package[?]. The script also sets maximum voltage and current levels to avoid damage to the phone during experimentation. Each measurement was performed for a user-specified duration of time, with a frequency of approximately 12 measurements per second.

3.2 Power Profiling

In order to determine the constants for the MDP equations, the first step is to determine the baseline average power consumption, when no user applications are running. To do this, all non-essential processes were killed, all radios were disabled and the display was turned off. With the phone in this idle state, measurements were taken in the manner described above. By averaging the data collected from this test, we derived the idle power consumption of 6.29 mW. Next, the power consumptions for voice call and WiFi connection were measured and compared to the baseline (idle) power consumption in order to determine their standalone power consumptions. In the case of the voice call, data was collected while an average level conversation was carried out over the cellular interface. WiFi and display were turned off, and all non-essential processes were disabled, as

in the baseline idle power consumption measurements. The average power consumption was calculated and the baseline power consumption was subtracted from it to get the result of 581.95 mW. Likewise, for the WiFi measurements, the display was off, only essential processes were running, and there was no network traffic except what is needed to maintain the connection. Figure 1 shows the data collected and used to determine the baseline power consumption for WiFi when it is on and connected to a network, but not explicitly sending or receiving any data. The average was 837.00 mW, while the power consumption of the WiFi alone was 830.71 mW. This clearly indicates the need for optimizing the power consumption of the WiFi radio that was discussed in the previous section.

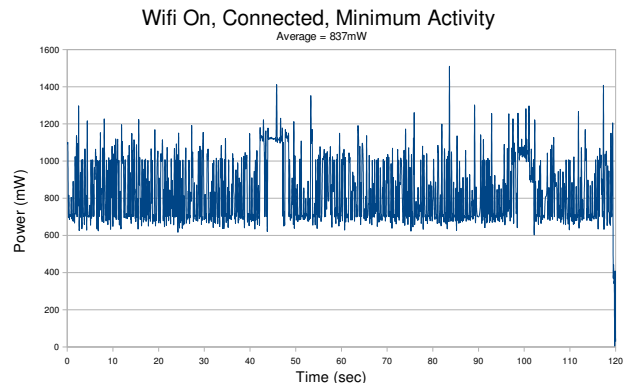


Figure 1: Minimum Activity while WiFi is On

To determine the average startup energy of the WiFi interface, a time measurement as well as a power measurement is necessary. We created a loop to turn the WiFi radio on and off and sampled these events to determine the average startup energy. The startup process begins when the WiFi is turned on from a previously off state, and continues while the power fluctuates until the WiFi is completely on and stabilized. This is shown in Figure 2 where the WiFi starts in an off state, is immediately turned on, and then completes five cycles of on and off until it once again ends in an off state. The average of each startup energy (the product of the duration of the startup and its average power consumption) was determined to be 647 J.

To find the power consumption of transmitting data over the WiFi interface, we wrote an application to run on the Android phone that creates and sends UDP packets of a given size. By running this application and taking power measurements as described before, the average power was found, and the baseline power consumptions for running the application and for WiFi on in its minimal state were subtracted to equal 395.70 mW. At this time, we also used the application to test the power consumption of transmitting data over the cellular interface (2253 mW), and determined that it is more power intensive than transmitting over the WiFi interface (1226 mW) as was expected.

We summarize the above measurement results in Table 2. These numbers are used in the MDP formulation to find optimal actions numerically.

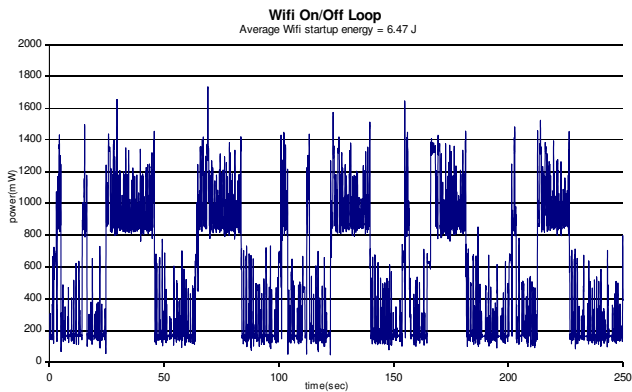


Figure 2: Measuring the Average Startup Energy of the WiFi Interface

| Constant | Energy (Joules) |
|----------|-----------------|
| e_c | 34.92 |
| e_s | 44.57 |
| e_w | 6.47 |
| e_m | 23.74 |
| e_I | 49.84 |

Table 2: Values for Defined Constants (unit time = 1 minute)

3.3 Voice Activity Profiling

A 66-day call log history was collected². The first 53 days of the history was used to generate the user profile, the later 13 days were used to run the simulation as 13 different test cases. Figure 3 shows the 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 unit time interval within a day. The probability of getting a phone call at a given time, $P_c(t)$, is thus estimated by dividing the frequencies of time interval t by the number of days the call log recorded.

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

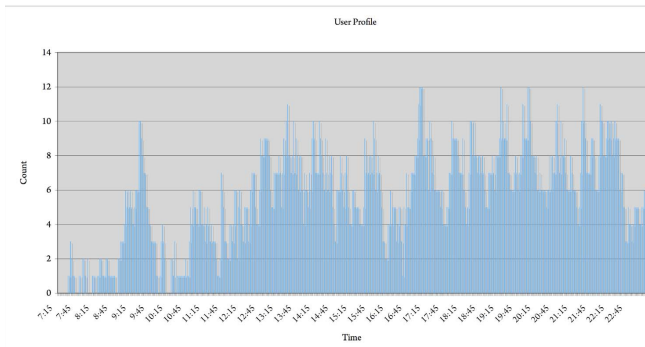


Figure 3: User Voice Communication Profile

Table 3: A day with light voice call

| Metric | 10 | 20 | 30 | 40 | 50 | 60 | MDP |
|------------|------|-------|-------|-------|-------|-------|-------|
| N_{syn} | 74 | 47 | 32 | 23 | 19 | 16 | 68 |
| Mean | 9.99 | 20.04 | 29.97 | 40.04 | 49.95 | 59.94 | 14.09 |
| Dev | 0.12 | 0.35 | 0.17 | 0.46 | 0.22 | 0.24 | 19.28 |
| M_{call} | 10 | 20 | 20 | 20 | 20 | 20 | 20 |
| T_{out} | 740 | N.A. | N.A. | N.A. | N.A. | N.A. | 959 |
| $E_r(T)$ | 0 | 105 | 180 | 225 | 245 | 260 | 0 |

4. RESULTS AND DISCUSSION

We implemented the MDP optimization framework for the email synchronization and WiFi radio control applications using the above power measurement data and user profile. The key results are presented as follows.

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 a decision table derived 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 10, 20, 30, 40, 50 and 60 minutes. The phone was simulated to discharge from time $t=0$ to time $t=960$ and no synchronization would be done during a phone call. We assumed that $f_r(E_r) = 0$; i.e., remaining energy at the phone charging time $t = 960$ has no value. Therefore, it is desirable to have the phone uses all its power by the end of $t = 959$ (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 HTC mobile phone running Android platform. The initial energy level is 400 units (around 133 minutes of talk time) (e.g., $E_r = 400$ at $t=0$). Energy consumption for each email synchronization (e_c) is 5 units. Energy consumption for making a phone call per minute (e_c) is 3 units. Each energy unit is approximately 11J.

We run the simulation in the later 13 days in the user call log (profile shown in Figure 3) as 13 different test cases. The reward function of email synchronization used was $\sqrt{\tau + 1}$. The entire discharging period was divided into 2-minute time intervals. We present the results in detail for three representative days with light, moderate, and heavy voice traffic, in Tables 3-5. Due to space constraints we omit the results for the other 10 days as they follow the same trend. In the results, 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 battery run out of energy, and $E_r(T)$ is the remaining energy at the end of the day when phone recharges.

Under our developed MDP email-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 which the phone did not power off due to insufficient energy before the charging time ($T = 960$), the number of synchronization made by our policy is always higher or equal to those of fixed-frequency policy.

Under our MDP policy, the phone ran out of battery only when the actual talk time in the test cases is close to the

Table 4: A day with moderate voice call

| Metric | 10 | 20 | 30 | 40 | 50 | 60 | MDP |
|------------|-------|-------|-------|-------|-------|-------|-------|
| N_{syn} | 73 | 46 | 31 | 23 | 18 | 15 | 46 |
| Mean | 10.03 | 20.70 | 30.71 | 41.39 | 51.78 | 60.07 | 20.85 |
| Dev | 0.29 | 4.38 | 3.73 | 6.12 | 7.58 | 0.57 | 25.41 |
| M_{call} | 13 | 58 | 58 | 58 | 58 | 58 | 58 |
| T_{out} | 733 | 953 | N.A. | N.A. | N.A. | N.A. | N.A. |
| $E_r(T)$ | 0 | 0 | 71 | 111 | 136 | 151 | 0 |

Table 5: A day with heavy voice call

| Metric | 10 | 20 | 30 | 40 | 50 | 60 | MDP |
|------------|-------|-------|-------|-------|-------|-------|--------|
| N_{syn} | 44 | 29 | 24 | 19 | 17 | 14 | 8 |
| Mean | 11.07 | 21.31 | 30.75 | 40.42 | 51.00 | 61.29 | 119.88 |
| Dev | 4.40 | 3.71 | 2.26 | 2.03 | 3.79 | 3.37 | 72.12 |
| M_{call} | 61 | 86 | 94 | 102 | 105 | 110 | 121 |
| T_{out} | 488 | 619 | 739 | 783 | 868 | 896 | N.A. |
| $E_r(T)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

maximum talk time supported by the battery. In the experiment, it happened in three days when there are over 100 phone-call minutes. In two cases, the battery life of the MDP policy is only outlived by that of the 60-minute policy, by 2 and 4 minutes, respectively. In the other case, the battery life of our policy outlived all others, by 159 minutes.

In fixed-frequency policy, the standard deviation of the synchronization period is non-zero because no email synchronization takes place when a phone call is taking place. Using our policy, the standard deviation of 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, 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. This is also reasonable because charging time may vary.

In summary, compared to the fix-frequency synchronization policy, our scheme is dynamic — it allows more synchronization when voice traffic volume is low; it reduces data service frequency when voice traffic is heavy. Because of this dynamic nature, it serves user more effective 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., $F_r(E_r) = \log(1 + E_r)$ at time T . This value will reward remaining energy at time T and makes data synchronization more conservative. Second, we will investigate the tradeoff between accuracy and table size. As discussed earlier, the size of the table is proportional to the number of states. If we set a time unit to be 5 minutes, then the size of the decision table will reduce by 60% at the cost of granularity. Last, we will investigate different reward functions ($R_s(\cdot)$) that can achieve different tradeoffs of voice communication and data synchronization services.

4.2 WiFi Interface Control

As reported in Section 3, there is an energy cost associated with waking up the WiFi radio and an energy cost associated with keeping the WiFi radio on, whether any data is transmitted or not. So, it makes sense to *aggregate* a bunch of (delay-tolerant) WiFi transmission requests, wake-up the WiFi radio, transmit all the aggregated requests and turn the WiFi radio off. The critical issue is when to turn on the WiFi radio, as it directly impacts the latency experienced by a transmission request which has to be minimized. The decision table is generated based on Eqs. 4 and 5, taking the user profile, the energy-related parameters of the phone as an input. The user profile provides the probability of a new WiFi transmission request at a given time, $p_c(t)$, and the expected charging time, T , of the phone.

We show a small part of the decision table in Table 6.

Table 6: WiFi interface decision table

| t | E_r | N | Current WiFi | Decision |
|-----|-------|-----|--------------|----------|
| 28 | 19 | 21 | OFF | ON |
| 28 | 19 | 22 | OFF | ON |
| 28 | 18 | 0 | OFF | OFF |
| 28 | 18 | 1 | OFF | OFF |
| 28 | 18 | 2 | OFF | OFF |

We compared the proposed MDP-based optimization policy with two simple policies described below:

1. *Once On, Always On policy:* In this policy, the WiFi radio is turned on at the first WiFi transmission request and remains on until the battery runs out. As a consequence, all subsequent requests (after the first one) are served immediately.
2. *On Demand policy:* In this policy, the WiFi is turned on by a WiFi transmission request. It remains on if there are pending requests. Otherwise it is turned off till the next request arrives. Therefore, in this case, requests that arrive after the WiFi is turned off, have to pay the energy cost of waking up the WiFi radio.

The phone was simulated to discharge from time $t=0$ to time $t=420$. At each 15-second time interval, only one new request is allowed to make. Seven test cases were used. Each test case has a different fixed probability of a new WiFi transmission request.

The reward function used in generating the decision table was $\log(N + 1)$. The following platform specific parameters were derived from the HTC mobile phone platform running Android, as described in Section 3. The initial energy level is 700 units. Energy consumption of turning on the WiFi interface (e_w) is 3 units. Energy consumption of keeping the WiFi interface on for 15 seconds (e_c) is 6 units. Energy consumption of transmitting data on WiFi for 5 seconds, in addition to e_c and e_m , is 1 unit. The WiFi interface takes 1 time unit (about 15 seconds) to wake up. Each energy unit is approximately 2 joules.

In the Tables 7, 8, 9, we report the performance of Always-On, On-demand, and our MDP policies. In the tables, the numbers 1 through 7 represent the different test cases which capture different transmission request probabilities (corresponding to different usage patterns), p is the corresponding

Table 7: Always ON WiFi Interface

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------------|-------|-------|-------|-------|-------|-------|--------|
| p | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 |
| N_{req} | 46 | 84 | 121 | 160 | 215 | 246 | 298 |
| N_{serve} | 12 | 22 | 35 | 42 | 51 | 60 | 70 |
| W | 0.083 | 0.045 | 0.028 | 0.024 | 0.020 | 0.017 | 0.0143 |
| Dev | 0.276 | 0.208 | 0.166 | 0.152 | 0.138 | 0.128 | 0.119 |
| T_{out} | 120 | 115 | 112 | 111 | 108 | 106 | 105 |

Table 8: On-demand WiFi interface control

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------------|-------|-------|-------|-------|-------|-------|-------|
| p | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 |
| N_{req} | 46 | 84 | 121 | 160 | 215 | 246 | 298 |
| N_{serve} | 46 | 83 | 89 | 92 | 99 | 102 | 108 |
| W | 0.848 | 0.686 | 0.584 | 0.522 | 0.434 | 0.333 | 0.278 |
| Dev | 0.359 | 0.464 | 0.493 | 0.499 | 0.496 | 0.471 | 0.448 |
| T_{out} | N.A. | 419 | 333 | 271 | 199 | 169 | 155 |

request probabilities, N_{req} is the total number of transmission requests, N_{serve} is the total number of flows served, W is the mean waiting time, Dev is its standard deviation, T_{out} is when the battery runs out, and $E_r(T)$ is the remaining energy at the end of the recharge time T specified by the user.

Clearly, as shown in Table 7, the *Always On* policy depletes power very quickly (and thus all following voice calls will be missed). It can serve only about 25% of the requests. As expected, when the radio is on, the delay and delay variance are negligible. The *On Demand* policy is somewhat better, based on the T_{out} value, and the number of requests serviced. Note that W decreases as p increases because more requests arrive when the WiFi radio is on as p increases. The MDP based policy clearly performs well both in terms of the T_{out} metric and in terms of the number of requests served (N_{serve}). In the simulation, the battery did not run out and almost all requests are served. However, as expected there is a price to pay in terms of additional delay as indicated by the waiting time metric W . In addition, as p increases, we can see that the average delay increases as well, which means that the policy accumulates more flows before transmission.

4.3 Discussion

A few remarks are in order related to the scalability and computation complexity of the proposed approach. First, we have the option of implementing the computational expensive algorithm on the phone or on a more powerful server through Internet connection. For instance, to generate the lookup table used in email synchronization, it takes sub-seconds 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 computation of the lookup table does not need to be frequent, say in once in a few days/weeks.

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) * 1byte. In other words, the size is reduced from 2.88MB to 48KB.

Table 9: MDP-based WiFi interface control

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------------|-------|-------|------|------|------|------|------|
| p | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 |
| N_{req} | 46 | 84 | 121 | 160 | 215 | 246 | 298 |
| N_{serve} | 46 | 84 | 121 | 159 | 213 | 243 | 285 |
| W | 0.804 | 0.881 | 2.12 | 2.38 | 3.01 | 3.50 | 3.71 |
| Dev | 0.397 | 0.521 | 2.25 | 2.33 | 2.59 | 2.74 | 2.65 |
| T_{out} | N.A. | N.A. | N.A. | N.A. | N.A. | N.A. | N.A. |

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 one, respectively. The performance degradation is minor and graceful.

Non-zero reward for remaining energy could be explored instead. We considered a non-zero reward function: $f(E_r) = c \log(1 + E_r)$, where c is a constant. The main effect 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 [?] were one of the first to use Markov decision processes to address power optimization policies for notebook or other battery-operated systems. Quality versus resource utilization trade-offs have been studied widely in the area of video streaming [?, ?, ?, ?, ?, ?]. In comparison, our work is more dynamic in nature, under the notion that we want to maximize user experience until the explicit charging time, instead of maximizing lifetime. For example, the optimal action to turn on/off disk will not change over time in [?], but will depend explicitly on the current time and remaining energy in this work.

In [?] 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 reach the next recharge time, which is the focus of this paper. Also, we address problems such as the WiFi radio optimization which is not addressed in [?].

CMU researchers studied OS support for resource scalable computation and energy aware adaptive computation [?, ?, ?]. In particular, in [?] the authors demonstrate a 30% extension in battery life through collaborative optimization of the operating systems and the application. Duke researchers [?] 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* the active applications. In [?], the authors propose a dynamic software management framework to improve battery life that is based on quality-of-service (QoS) adap-

tation and user-defined priority. In comparison, the goal of their works is to extend the battery lifetime by limiting the average discharge rate. In addition, their focus/target is a general purpose notebook computer, as opposed to a mobile phone with voice communications as its primary functionality. In the area of sensor networks, UCLA researchers [?] discuss scheduling tasks to accommodate the constraints of energy harvested from the environment such as solar panels. In [?], an alternative approach to reactive optimization is discussed. The main difference between the related works listed here and the work proposed in this paper 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 AND FUTURE WORK

In this paper we proposed a general mathematical framework to optimize software on mobile phones using Markov decision process. We developed techniques to reduce the table size for certain applications like data synchronization. We argued that on a mobile phone, talk-time optimization should be the primary goal and it should be a user-defined parameter, as it depends on the usage pattern (when a phone is recharged) that varies from one individual to another. This makes the problem different from the energy minimization work in embedded software such as video streaming on battery-operated notebooks/handheld devices that is primarily driven by extending the battery-life with minimal impact on quality. Though there is an implicit time constraint in these problems as well, it is derived from the workload (such as time to process a frame) as opposed to an external global time constraint for all the applications. Future work would include extending the WiFi radio control to general radio selection on a mobile phone given that most smartphones have multiple radios. In addition, the current scheme optimizes one application at a time. Joint power management of multiple applications will be considered in our future work. The techniques presented in this paper currently focus on applications only. So, in the future we will explore collaborative optimization between the operating system and applications.

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

7. APPENDIX

7.1 Proof of Theorem 1

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

$$V(t, E_r, \tau + \epsilon) - V(t, E_r, \tau) \leq R_s(\tau + \epsilon) - R_s(\tau), \quad (6)$$

for $\epsilon \geq 0$.

Proof: The property can be proved using backward induction. Because of the boundary condition in Eq. 2, we have

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

Therefore, Eq. 6 holds for $t = T$. We next use backward induction. Assume Eq. 6 holds for $t + 1, t + 2, \dots, T$, we need to prove it holds for t .

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

$$\begin{aligned} & V(t, E_r, \tau + \epsilon) - V(t, E_r, \tau) \\ &= p_c(t) (v_c(t, E_r, \tau + \epsilon) - v_c(t, E_r, \tau)) \\ &\quad + (1 - p_c(t)) \max \{v_s(t, E_r, \tau + \epsilon), v_i(t, E_r, \tau + \epsilon)\} \\ &\quad - (1 - p_c(t)) \max \{v_s(t, E_r, \tau), v_i(t, E_r, \tau)\} \\ &\stackrel{(1)}{\leq} p_c(t) (v_c(t, E_r, \tau + \epsilon) - v_c(t, E_r, \tau)) \\ &\quad + (1 - p_c(t)) \max \{v_c(t, E_r, \tau + \epsilon) - v_c(t, E_r, \tau), \\ &\quad v_i(t, E_r, \tau + \epsilon) - v_i(t, E_r, \tau)\} \end{aligned}$$

where (1) holds because

$$\begin{aligned} & \max(a, b) - \max(c, d) \\ &= \max(a - \max(c, d), b - \max(c, d)) \\ &\leq \max(a - c, b - d). \end{aligned}$$

We consider the first term next.

$$\begin{aligned} & v_c(t, E_r, \tau + \epsilon) - v_c(t, E_r, \tau) \\ &= E_{L_c} [V(t + L_c, (E_r - L_c * e_c)^+, \tau + \epsilon + L_c) \\ &\quad - V(t + L_c, (E_r - L_c * e_c)^+, \tau + L_c)] \\ &\stackrel{(2)}{\leq} E_{L_c} [R(\tau + \epsilon + L_c) - R(\tau + L_c)] \\ &\stackrel{(3)}{\leq} E_{L_c} [R(\tau + \epsilon) - R(\tau)] \\ &= R(\tau + \epsilon) - R(\tau) \end{aligned}$$

where (2) holds by the hypothesis, and (3) holds because $R(\cdot)$ is a subadditive increasing function.

Consider the second term.

$$\begin{aligned} & \max \{v_c(t, E_r, \tau + \epsilon) - v_c(t, E_r, \tau), \\ & v_i(t, E_r, \tau + \epsilon) - v_i(t, E_r, \tau)\} \\ &\leq \max \{V(t + 1, (E_r - e_s)^+, 1) + R_s(\tau + \epsilon) \mathbf{1}\{E_r \geq e_s\} \\ &\quad - V(t + 1, (E_r - e_s)^+, 1) + R_s(\tau) \mathbf{1}\{E_r \geq e_s\}, \\ &\quad V(t + 1, E_r, \tau + \epsilon + 1) - V(t + 1, E_r, \tau + 1)\} \\ &\leq \max \{R_s(\tau + \epsilon) - R_s(\tau), R_s(\tau + \epsilon + 1) - R_s(\tau + 1)\} \\ &\leq R_s(\tau + \epsilon) - R_s(\tau). \end{aligned}$$

Combining the above two results, we have

$$V(t, E_r, \tau + \epsilon) - V(t, E_r, \tau) \leq R_s(\tau + \epsilon) - R_s(\tau).$$

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

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

$$v_s(t, E_r, \tau) \geq v_i(t, E_r, \tau). \quad (7)$$

If $E_r < e_s$, then

$$v_s(\tau) = V(t + 1, (E_r - e_s)^+, 1) + R_s(\tau) \mathbf{1}\{E_r \geq e_s\} = 0.$$

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

$$\begin{aligned} v_s(t, E_r, \tau) &= V(t + 1, E_r - e_s, 1) + R_s(\tau) \\ &= V(t + 1, E_r - e_s, 1) + R_s(\tau^*) + R_s(\tau) - R_s(\tau^*) \\ &= v_s(\tau^*) + R_s(\tau) - R_s(\tau^*) \\ &\stackrel{(4)}{\geq} v_i(\tau^*) + R_s(\tau) - R_s(\tau^*) \end{aligned}$$

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

$$\begin{aligned} v_s(t, E_r, \tau) &= V(t+1, E_r, \tau+1) \\ &\stackrel{(6)}{\leq} V(t+1, E_r, \tau^*+1) + R_s(\tau+1) - R_s(\tau^*+1) \\ &= v_i(\tau) + R_s(\tau+1) - R_s(\tau^*+1) \\ &\stackrel{(7)}{\leq} v_i(\tau^*) + R_s(\tau) - R_s(\tau^*) \end{aligned}$$

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 $\tau \geq \tau^*$.